The Modified Network

```
Process q3(in int a, in int b, out int x) {
int k; bool sw = true;
loop
    if sw then
        k = a.receive() on timeout(d) do
            sw = !sw;
            continue;
    else
        k = b.receive() on timeout(d) do
            sw = !sw;
            continue;
    end if;
    x.send(k);
    sw = !sw;
end loop; }
```

- Consider q3 instead of p3:
  - Process q3 first tries channel *a* or *b*, depending on sw, like in the previous version.
  - But, instead of blocking, if nothing comes after a timeout *d*, q3 will switch to read a token from the other channel.

- With q3 we do not have a Kahn process network.
- The system is not deterministic.

هذه الشريحة تسلط الضوء على تعديل بسيط في سلوك عملية واحدة (p3) وكيف أن هذا التعديل - البسيط في مظهره - له تأثير جذري على خاصية أساسية من خصائص شبكات Kahn Process Networks وهي **الحتمية**.

منطق العملية q3 : لنتفحص الكود الخاص بها:

```
Process q3(in int a, in int b, out int x) {
int k; bool sw = true;
loop {
if sw then
k = a.receive() on timeout(d) do   // حاول القراءة من 'a' (C3) ولكن بمهلة زمنية 'd'
    sw = !sw;        // إذا انتهت المهلة ولم تصل بيانات، اعكس sw
    continue;        // انتقل للتكرار التالي في الحلقة
else
k = b.receive() on timeout(d) do   // إذا sw خاطئ، حاول القراءة من 'b' (C4) بمهلة زمنية 'd'
    sw = !sw;        // إذا انتهت المهلة، اعكس sw
    continue;        // انتقل للتكرار التالي
end if;
// إذا وصلنا هنا، فهذا يعني أن القراءة تمت بنجاح قبل انتهاء المهلة
x.send(k);  // أرسل القيمة المستلمة إلى المخرج 'x' (O)
```

sw = !sw; // اعكس قيمةsw (هذا يحدث فقط عند القراءة الناجحة)

}}

المقارنة مع p3 الأصلية :المنطق الأساسي لا يزال محاولة القراءة بالتناوب من a (C3) و b (C4) بناءً على قيمة sw **لكن الاختلاف الجوهري** هو استخدام on timeout(d) do هذا يعني أن العملية q3 لا تنتظر إلى الأبد (Blocking Read) إذا كانت القناة فارغة بل تنتظر لفترة زمنية محددة d

- إذا وصلت البيانات خلال المهلة d يتم استلامها، إرسالها إلى المخرج، ثم عكس sw، وتستمر الحلقة.
- إذا انتهت المهلة d ولم تصل بيانات، فإن القراءة تفشل (لا يتم استلام قيمة)، يتم عكس sw، ويتم استخدام continue للقفز مباشرة إلى بداية التكرار التالي في الحلقة، دون محاولة الإرسال إلى x.send(k) هذا يعني أن q3 ستحاول القراءة من القناة الأخرى في التكرار التالي.

- **"Process q3 first tries channel a or b, depending on sw, like in the previous version."** لا تزال q3 تحاول القراءة بالتناوب بناءً على sw مثل p3 الأصلية.
- **"But, instead of blocking, if nothing comes after a timeout d, q3 will switch to read a token from the other channel."**

هذه هي النقطة الحاسمة. بدلاً من الانحصار والانتظار إلى أن تتوفر البيانات (السلوك القياسي فيKPN)، إذا لم تصل البيانات خلال فترة المهلة d، فإن " q3 تستسلم" من القراءة الحالية، تعكس مؤشر التناوب sw، وتحاول القراءة من القناة الأخرى في المحاولة التالية.

- **"With q3 we do not have a Kahn process network."**

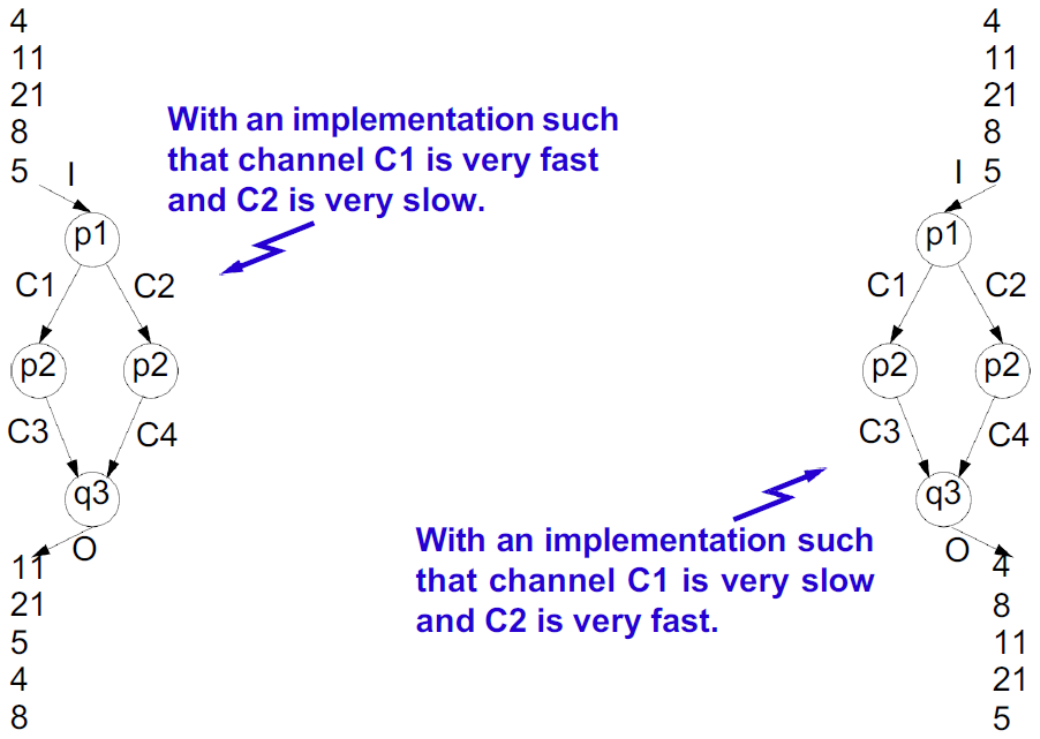
هذا هو الاستنتاج الهام الأول. بمجرد استبدال p3 بعملية q3 التي تستخدم القراءة بمهلة زمنية(timeout) ، فإن الشبكة الناتجة **لم تعد تعتبر شبكة عمليات كان (KPN) بالمعنى الدقيق للمصطلح** .السبب هو كسر إحدى القواعد الأساسية لـ KPN وهي القراءة المانعة التي لا تعتمد على التوقيت.

- **"The system is not deterministic."**هذا هو الاستنتاج الأهم والنتيجة المباشرة لعدم كونها KPN الشبكة المعدلة **ليست حتمية**.

لماذا يؤدي timeout(d) إلى فقدان الحتمية؟

لنتذكر أن الحتمية في KPN تعتمد على أن سلوك النظام يتحدد فقط بمحتوى القنوات (التوكنات) ومنطق العمليات، وليس بالتوقيت . القراءة المانعة تضمن أن العملية تنتظر بصبر إلى أن تتوفر البيانات اللازمة، بغض النظر عن سرعة العملية التي تنتج هذه البيانات.
عندما نستخدم timeout(d)، فإن سلوك العملية q3 يصبح **معتمداً على التوقيت** .إذا وصلت البيانات إلى C3 مثلاً قبل انتهاء المهلة d، فإن q3 ستقرأها وترسلها. ولكن إذا تأخر وصول البيانات إلى C3 قليلاً وتجاوز المهلة d (لأن العملية p2a كانت بطيئة في ذلك التكرار مثلاً)، فإن q3 لن تقرأ من C3 في تلك المحاولة، بل ستقلب sw وتحاول القراءة من C4.

The Modified Network



لنتخيل سيناريوين لنفس حالة القنوات الأولية:

- **السيناريو1:** p2a تعمل بسرعة، بيانات تصل إلى C3 قبل انتهاء مهلة q3. q3 تقرأ من C3، ثم تحاول القراءة من C4.
- **السيناريو2:** p2a تعمل ببطء أكبر (ربما بسبب مقاطعة في النظام المدمج)، بيانات تتأخر في الوصول إلى C3 وتصل بعد انتهاء مهلة q3. d تفشل في القراءة من C3، تقلب sw، وتحاول القراءة من C4 أولاً في المحاولة التالية.

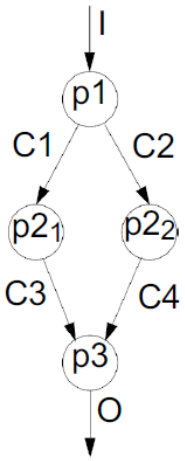
في هذين السيناريوين، مع نفس المدخلات ونفس منطق العمليات، تسلسل قراءات q3 من C3 و C4 سيختلف اعتماداً على التوقيت الدقيق لوصول البيانات. وبما أن مخرجات q3 تعتمد على ما تقرأه من أي قناة وفي أي ترتيب، فإن تسلسل المخرجات النهائية على O يمكن أن يتغير " **هذا هو فقدان الحتمية** ".

ربط هذا بالروبوتات:

في الأنظمة المدمجة للروبوتات، التعامل مع التوقيت أمر واقعي وضروري (مثلاً، قراءة حساس يجب أن تتم خلال فترة زمنية معينة). ولكن إدخال اعتمادية صريحة على التوقيت في منطق الاتصال بين العمليات بهذا الشكل (مثل القراءة بمهلة زمنية مع تغيير السلوك عند انتهاء المهلة) يجب أن يتم بحذر شديد، لأنه يجعل التحقق من صحة النظام واختباره أصعب بكثير. يصبح سلوك الروبوت حساساً لتغيرات بسيطة في سرعة المعالج أو أوقات الوصول إلى الذاكرة أو المقاطعات، مما قد يؤدي إلى سلوك غير متوقع في ظروف تشغيل مختلفة.

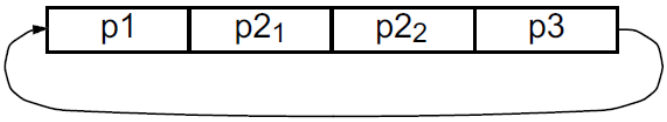
إذاً، حتى تعديل بسيط مثل إضافة مهلة زمنية للقراءة يمكن أن يكسر خاصية الحتمية الأساسية لنموذج KPN ، مما يجعل النظام الناتج أكثر تعقيداً وغير قابل للتنبؤ بنفس الدرجة. الحفاظ على الحتمية في KPN يتطلب الالتزام الصارم بقواعده، خاصة القراءة المانعة النقية.

Scheduling of Kahn Process Networks



- Let us imagine we have to implement the system on a single processor architecture.

Let's try the following static schedule:



The system will block!

هذه الشريحة تتناول تحدياً عملياً يواجهنا عند تطبيق نماذج مثل Kahn Process Networks (KPN) على أنظمة حقيقية، وهو **جدولة شبكات عمليات كان(Scheduling of Kahn Process Networks)** ، خصوصاً عند التنفيذ على **معالج واحد(single processor)** .

إن النظام المدمج غالباً ما يحتوي على معالج مركزي واحد. بينما في نموذج KPN نرسم العمليات كدوائر مستقلة يمكنها نظرياً العمل بالتوازي. في الواقع على معالج واحد، لا يمكن تنفيذ سوى عملية واحدة في لحظة زمنية معينة. هنا يأتي دور **الجدولة(Scheduling)** ، وهي عملية تحديد أي عملية ستعمل ومتى.

هذه الشريحة تتناول هذا التحدي وتعرض حلاً بسيطاً جداً (وهو غالباً غير فعال) وتوضح عيوبه.

- نظام KPN النظري مثالي لنمذجة التوازي، لكن تطبيقه على معالج واحد يتطلب آلية لتوزيع وقت المعالج على العمليات المختلفة.
- **الاقتراح** :هنا يتم اقتراح طريقة جدولة بسيطة جداً :الجدولة **الساكنة (Static Schedule)** . في الجدولة الساكنة، يتم تحديد تسلسل ثابت ومسبق للعمليات التي سيحاول المعالج تنفيذها، وهذا التسلسل يتكرر باستمرار.
- **رسم الجدول الساكن** :يوضح الرسم تسلسل الجدولة المقترح وهو :p1 ثم p2 (النسخة الأولى) ثم p2 (النسخة الثانية) ثم p3.ثم يتكرر هذا التسلسل في حلقة لا نهائية ... -> p3 -> p2b -> p2a -> p1 -> p3 -> p2b -> p2a -> p1 .(لقد افترضنا أن الـ p2 الأولى في الجدول هي p2a التي تقرأ من C1 ، والثانية هي p2b التي تقرأ من C2 ، بناءً على الترتيب المنطقي للشبكة).
- **النتيجة "The system will block!"** :هذه هي الخلاصة الصادمة لهذا الجدول المحدد. النظام باستخدام هذا الجدول الساكن **سينحصر(block)** ، أي سيتوقف عن العمل ولن يحرز أي تقدم إضافي بعد نقطة معينة.

لماذا يحدث الانحصار (Blocking) مع هذا الجدول؟

لنتذكر خاصية القراءة المانعة (Blocking Read) في KPN .العملية التي تحاول القراءة من قناة فارغة تتوقف وتنتظر.

سنتتبع الجدولة في الخطوات الأولى ونرى ماذا يمكن أن يحدث:

- **الخطوة 1: p1 تعمل** .إذا كان المدخل I يحتوي على بيانات فإن p1 تقرأها، تعالجها (زوجي C1 <- ، فردي C2 <-)، وترسلها. الإرسال غير مانع .p1 تنتهي من دورها وتنتظر دورها التالي.
- **الخطوة 2: p2a تعمل** .تحاول القراءة من C1. إذا كانت p1 قد أرسلت إليها توكن في الخطوة السابقة (لأن المدخل كان زوجياً)، p2a تقرأ التوكن، تمرره إلى C3، وتنهي دورها. إذا كانت C1 فارغة (لأن المدخل كان فردياً ولم يتم إرسال شيء إلى C1 ، أو لم يكن هناك مدخل أصلاً في I لـ p1 لتعالجه)، فإن p2a ستحاول القراءة من C1 وتجدها فارغة، وبالتالي **ستنحصر (block)** هنا.
- **الخطوة 3: p2b تعمل** .تحاول القراءة من C2. إذا كانت p1 قد أرسلت إليها توكن (لأن المدخل كان فردياً)، p2bتقرأه، تمرره إلى C4، وتنهي دورها. إذا كانت C2 فارغة، p2b **ستنحصر**.
- **الخطوة 4: p3 تعمل** .تحاول القراءة من مدخلاتها(C3 و C4) بناءً على منطقها p3 (والذي رأينا أنه معقد في المثال السابق)، لكن حتى لو كان التناوب البسيط(فإنها تحتاج لتوكنات من C3 و C4 أو على الأقل التوكن الذي تحاول قراءته في دورها الحالي من إحدى القناتين). هذه التوكنات يجب أن تأتي من p2a و p2b

لماذا يؤدي هذا إلى الانحصار الكامل؟

إذا وصلت الجدولة إلى p3 و حاولت p3 القراءة من C3 أو C4 ووجدت القناة فارغة، فإن p3 **ستنحصر** .بما أن هذا يتم على معالج واحد، فإن انحصار p3 يعني أن **المعالج بأكمله ينحصر**، حيث لا يمكن لأي عملية أخرى أن تعمل، بما في ذلك p1, p2a, p2b التي قد تكون قادرة على إنتاج التوكن الذي تنتظره p3

يصبح لدينا حلقة مفرغة: p3 تنتظر توكنات من p2a و p2b، ولكن p2a و p2bلا يمكنهما العمل لإنتاج التوكنات لأن المعالج محتكر بواسطة p3 المنحصرة.

هذا هو جوهر مشكلة الجدولة الساكنة غير الفعالة في KPNs .الجدول يفرض ترتيباً لا يأخذ في الاعتبار جاهزية البيانات. إذا تم جدولة عملية مستهلكة (مثل p3) قبل أن تتمكن العمليات المنتجة (مثل p1 أو p2) من وضع البيانات التي تحتاجها في القنوات، فإن المستهلك سينحصر ويوقف النظام بأكمله.

الحل (بشكل عام):

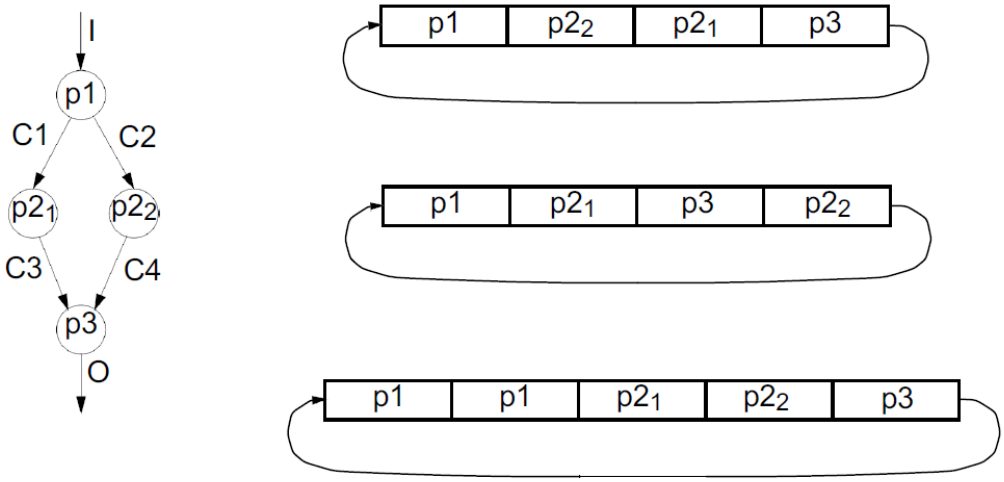
الحل لهذه المشكلة عادة ما يكون باستخدام **جدولة ديناميكية (Dynamic Scheduling)** . في الجدولة الديناميكية (مثل تلك التي تستخدمها أنظمة التشغيل)، عندما تحاول عملية ما القراءة وتجد القناة فارغة فتنحصر، يقوم المجدول بسحب المعالج منها وإعطائه لعملية أخرى تكون جاهزة للتنفيذ (أي لا تحاول القراءة من قناة فارغة). هذا يسمح للعمليات المنتجة بأن تعمل وتنتج التوكنات، مما قد يحرر في النهاية العملية المنحصرة.

ربط هذا بالروبوتات:

عندما نصمم برمجيات روبوتات معقدة كشبكة من المهام المتوازية (وهو أمر شائع جداً)، يجب أن نفكر في آلية الجدولة التي سنستخدمها على المعالج المدمج. الجدولة الساكنة البسيطة مثل هذه قد لا تكون مناسبة لأنها يمكن أن تؤدي إلى جمود (deadlock) وتوقف النظام. يجب استخدام آليات جدولة أكثر تطوراً تضمن عدم انحصار العمليات الحيوية التي تنتج البيانات التي تحتاجها أجزاء أخرى من النظام.

Scheduling of Kahn Process Networks

And all other schedules will block:



سوف نكمل الحديث عن **جدولة شبكات عمليات كان (Scheduling of Kahn Process Networks)** على معالج واحد، ونستعرض أمثلة لجداول ساكنة (Static Schedules) أخرى لنفس الشبكة.

1. العبارة **"And all other schedules will block:"** :هذه عبارة تدعي أن جميع الجداول الساكنة الممكنة لهذه الشبكة **ستؤدي إلى الانحصار(blocking)**
2. ثلاثة أمثلة إضافية لجداول ساكنة مقترحة، كل منها يمثل تسلسلاً مختلفاً لمحاولة تنفيذ العمليات، ويتكرر هذا التسلسل في حلقة:
   - الجدول 1 p1 | p2_2 | p2_1 | p3
   - الجدول 2 p1 | p2_1 | p3 | p2_2
   - الجدول 3 p1 | p1 | p2_1 | p2_2 | p3

لماذا كل الجداول الأخرى ستنحصر؟

الادعاء بأن "جميع" الجداول الساكنة ستنحصر قد يبدو مفاجئاً. ولكن بناءً على فهمنا لعمل هذه الشبكة وتحديات الجدولة الساكنة، يمكننا أن نرى الأساس لهذا الادعاء.

لنتذكر أن المشكلة الرئيسية في الجدولة الساكنة لشبكات KPN هي **ضمان أن العملية القارئة لا تحاول القراءة من قناة فارغة وتتسبب في انحصار المعالج قبل أن تتاح للعمليات المنتجة فرصة لوضع البيانات في تلك القناة.**

في شبكتنا:

- p1 هي المنتج الأولي للبيانات الداخلية (تنتج لـ C1 و C2).
- p2_1 (C1->C3) و p2_2 (C2->C4) هما منتجان لـ p3 ومستهلكان لـ p1
- p3 هي المستهلك النهائي للبيانات الداخلية (تستهلك من C3 و C4)

أي جدول ساكن سيعاني من خطر الانحصار إذا لم يكن قادراً على ضمان تدفق البيانات بشكل مستمر. هذا يتطلب نوعاً من التوازن الدقيق في ترتيب تنفيذ العمليات المنتجة والمستهلكة.

لنتأمل لماذا الأمثلة الإضافية المقدمة قد تنحصر أيضاً:

- **الجدول1 p1 | p2_2 | p2_1 | p3 :**
  - p1 تعمل، تنتج توكنات لـ C1 أو C2
  - p2_2 تعمل. إذا كان C2 فارغاً (لأن p1 أرسلت إلى C1 مثلاً)، p2_2 ستنحصر. حتى لو أرسلت p1 إلى C2 ، p2_2 تأخذها.
  - p2_1 تعمل. تحاول القراءة من C1 . إذا كانت C1 فارغة، ستنحصر.
  - p3 تعمل. تحاول القراءة من C3 و/أو C4 . إذا كانت أي منهما فارغة، p3 ستنحصر.
  - لنتخيل سيناريو حيث المدخلات لـ p1 تأتي ببطء نسبياً. قد تعمل p1 وتنتج توكن واحد. ثم تحاول p2_2 القراءة، تنجح أو تنحصر. ثم p2_1 تحاول، تنجح أو تنحصر. ثم p3 تحاول، تنجح أو تنحصر. إذا انحصرت p3 ، فإنها تمنع p1, p2_1, p2_2من العمل مرة أخرى لإنتاج البيانات التي تنتظرها. نفس المشكلة التي رأيناها سابقاً.
- **الجدول2 p1 | p2_1 | p3 | p2_2 :**
  - p1 تعمل.
  - p2_1 تعمل. تحاول القراءة من C1 .إذا كان C1 فارغاً، تنحصر.
  - p3 تعمل. تحاول القراءة من C3 و/أو C4 . إذا كانت أي منهما فارغة، تنحصر. هنا الخطر أكبر، لأن p3 تأتي مباشرة بعد p2_1 وقبل p2_2 .إذا كانت تحتاج بيانات من C4 التي تنتجها p2_2 ، فقد لا تكون p2_2 قد حصلت على فرصتها للعمل بعد.
  - p2_2 تعمل. تحاول القراءة من C2 .إذا كان C2 فارغاً، تنحصر.
- **الجدول3 p3 | p2_2 | p2_1 | p1 | p1 :**
  - هذا الجدول يعطي p1 فرصتين متتاليتين للعمل. هذا قد يساعد في ملء القناتين C1 و C2 بشكل أسرع في البداية.
  - p1 (المرة الأولى) تعمل.
  - p1 (المرة الثانية) تعمل.
  - p2_1 تعمل. تحاول القراءة من C1 .احتمال أن تجد بيانات في C1 أعلى بعد عمل p1 مرتين.
  - p2_2 تعمل. تحاول القراءة من C2. احتمال أن تجد بيانات في C2 أعلى.
  - p3 تعمل. تحاول القراءة من C3 و/أو .C4 بالرغم من أن C1 و C2 قد تكون امتلأت، إلا أن C3 و C4 تمتلئان فقط عندما تعمل p2_1 و p2_2 وتمرران البيانات. إذا كانت كمية البيانات التي تضعها p1 كبيرة جداً لدرجة أن C1 و C2 تظل ممتلئة، فإن p2_1 و p2_2 سيجدان بيانات للعمل عليها. ولكن بمجرد أن تبدأ p3 في الاستهلاك من C3 و C4 ، يمكن أن

يصل الحال إلى نقطة تحاول فيها p3 القراءة وتجد القناة فارغة لأن p2_1 أو p2_2 لم تلحق بالإنتاج في دورها السابق، وبالتالي تنحصر p3 ويقف النظام.

لماذا "كل" الجداول الساكنة؟

الادعاء بأن جميع الجداول الساكنة ستنحصر هو ادعاء قوي ويشير إلى وجود مشكلة أساسية في جدولة هذا النوع من الهياكل الذي يحتوي على التقاء بيانات - join - في p3 حيث تحتاج بيانات من مسارين مختلفين (باستخدام جدول ساكن بحت).

لكي لا ينحصر نظام KPN بجدول ساكن، يجب أن يكون الجدول قابلاً للإطلاق (Schedulable) أو دوري (Periodic) هذا يعني أنه يجب أن يكون هناك تسلسل ثابت من مرات إطلاق (firings) العمليات يضمن أن أي عملية تحاول القراءة ستجد دائماً بيانات في القناة التي تحاول القراءة منها في لحظة القراءة المجدولة تلك. العثور على مثل هذا الجدول لأنظمة معقدة ليس بالأمر السهل، وفي بعض الهياكل (مثل التي تحتوي على حلقات تغذية راجعة معقدة أو نقاط التقاء بيانات تتطلب مزامنة)، قد لا يوجد جدول ساكن يمنع الانحصار تماماً على معالج واحد بدون سعة قنوات لا نهائية (وهو افتراض نظري).

هذه الشريحة تسلط الضوء على أن الجدولة الفعالة لشبكات KPN على معالج واحد هي مهمة معقدة، وأن الحلول الساكنة البسيطة غالباً ما تفشل وتؤدي إلى جمود النظام بسبب طبيعة القراءة المانعة. هذا يدفعنا نحو الحاجة إلى جدولة ديناميكية أو هياكل شبكات أبسط أو استخدام معالجات متعددة.

ربط هذا بالروبوتات:

إن تصميم برمجيات الروبوتات كنظام من المهام المتوازية يتطلب فهماً عميقاً لكيفية تفاعل هذه المهام وكيف سيتم جدولتها على الأجهزة المتاحة. اختيار بنية الشبكة وخوارزمية الجدولة أمر حيوي لتجنب الجمود وضمان الاستجابة في الوقت الحقيقي. الجداول الساكنة قد تكون مناسبة جداً لمهام بسيطة ومتتابعة معروفة مسبقاً، لكن للهياكل المعقدة وديناميكية تدفق البيانات غير المتوقعة بالكامل (بناءً على المدخلات البيئية مثلاً)، قد تكون الجدولة الديناميكية أكثر ملاءمة ومرونة، رغم تعقيدها الإضافي.

Scheduling of Kahn Process Networks

- **Kahn process networks are *dynamic* dataflow models: their behavior is data dependent; depending on the input data one or the other process is activated.**
- **Kahn process networks cannot be scheduled statically ⇒ It is not possible to derive, at compile time, a sequence of process activations such that the system does not block under any circumstances.**

⬇

Kahn process networks have to be scheduled dynamically ⇒ which process to activate at a certain moment has to be decided, during execution time, based on the current situation.

⬇

There is an overhead in implementing Kahn process networks.

هذه الشريحة تلخص وتؤكد الاستنتاجات الهامة التي توصلنا إليها في الشرائح السابقة بخصوص **جدولة شبكات عمليات كان(KPN).**

1. **"Kahn process networks are dynamic dataflow models: their behavior is data dependent; depending on the input data one or the other process is activated."**
   - هذه النقطة تصف طبيعة KPN بأنها نماذج تدفق بيانات **ديناميكية** ."ديناميكية" هنا لا تعني أنها غير حتمية، بل تعني أن سلوكها (أي، أي عملية يمكنها التنفيذ ومتى) يعتمد على **البيانات المتاحة** في القنوات. العملية لا يمكنها التنفيذ إلا إذا كانت البيانات التي تحاول قراءتها متوفرة. إذا توفرت البيانات في قناة A وليس في قناة B، فإن العملية التي تنتظر بيانات من A هي التي يمكنها التقدم. هذا يجعل التنفيذ يعتمد على تدفق البيانات الفعلي.
2. **"Kahn process networks cannot be scheduled statically ⇒ It is not possible to derive, at compile time, a sequence of process activations such that the system does not block under any circumstances."**
   - هذه هي النقطة الجوهرية التي استخلصناها من الشرائح السابقة. بسبب طبيعة القراءة المانعة واعتماد سلوك النظام على البيانات الواردة (والتي لا تكون معروفة بالكامل وقت كتابة الكود أو الترجمة- compile time)، **لا يمكن بشكل عام** إنشاء جدول ساكن (Static Schedule) ثابت يضمن أن النظام **لن ينحصر أبداً** مهما كانت تسلسلات المدخلات أو التوقيتات النسبية لوصول البيانات (ما لم يكن لدينا سعة قنوات لا نهائية، وهو افتراض نظري).
   - وهنا نخلص إلى النتيجة المنطقية: بما أننا لا نعرف مسبقاً متى ستتوفر البيانات في أي قناة، فلا يمكننا تحديد تسلسل ثابت لتشغيل العمليات يضمن أن العملية التي ستعمل في دورها ستجد دائماً البيانات التي تحتاجها ولن تنحصر.
3. **"Kahn process networks have to be scheduled dynamically ⇒ which process to activate at a certain moment has to be decided, during execution time, based on the current situation."**

- هذا هو الاستنتاج الثاني الهام والحل العملي للمشكلة. بما أن الجدولة الساكنة غير ممكنة بشكل عام لتجنب الانحصار، فإن شبكات KPN (أو على الأقل تلك التي تحتوي على هياكل معقدة مثل نقاط الالتقاء) **يجب جدولتها ديناميكياً**.
- "ديناميكياً" هنا تعني أن قرار تشغيل أي عملية في أي لحظة يتم اتخاذه **أثناء وقت التنفيذ(execution time)** يراقب المجدول (Scheduler) حالة القنوات (هل تحتوي على بيانات؟) وحالة العمليات (هل تحاول القراءة؟ هل هي جاهزة للإرسال؟)، ويقرر بناءً على هذه الحالة الديناميكية أي عملية يجب أن تحصل على وقت المعالج بعد ذلك.
- الفكرة هي: إذا حاول المجدول تشغيل عملية ووجدت أنها تحتاج للقراءة من قناة فارغة، فبدلاً من الانحصار، يتم وضع هذه العملية في حالة انتظار ويقوم المجدول باختيار عملية أخرى جاهزة للتنفيذ (عملية يمكنها إحراز تقدم، مثل عملية منتجة لديها مدخلات لتتم معالجتها أو لديها بيانات جاهزة للإرسال). هذا يسمح للنظام بالاستمرار في العمل طالما أن هناك بيانات تتدفق أو عمليات يمكنها معالجة البيانات المتاحة.

4. **"There is an overhead in implementing Kahn process networks."**

- هذه هي التكلفة المترتبة على الحاجة للجدولة الديناميكية. تنفيذ مجدول ديناميكي يتطلب وقتاً من المعالج لمراقبة حالة القنوات والعمليات واتخاذ قرارات الجدولة في وقت التشغيل. هذا يمثل **عبئاً إضافياً (overhead)** مقارنة بنظام يمكن جدولته ساكنة (حيث يتم تحديد كل شيء مسبقاً).
- في الأنظمة المدمجة التي غالباً ما تكون مواردها محدودة (قدرة معالجة، ذاكرة)، يجب الموازنة بين فوائد نموذج KPN (الحتمية عند استخدام جدولة ديناميكية) وتكلفة هذا العبء الإضافي للجدولة الديناميكية.

ربط هذا بالروبوتات:

هذه الشريحة تلخص تحدياً هاماً في تطبيق النماذج المتوازية على أنظمة معالج واحد مثل معالجات الروبوتات المدمجة:

- لا يمكننا دائماً الاعتماد على ترتيب تنفيذ ثابت ومسبق للوظائف (قراءة حساس، معالجة، تحكم بمحرك).
- يجب أن يكون نظام التشغيل أو المجدول في الروبوت ذكياً بما يكفي لجدولة المهام بناءً على توفر البيانات (هل وصلت قراءة الحساس الجديدة؟ هل انتهت عملية معالجة الصورة؟).
- هذه الجدولة الديناميكية تأتي مع تكلفة يجب أخذها في الاعتبار عند تصميم النظام واختيار الأجهزة والبرمجيات.

الخلاصة هي أن نموذج KPN يقدم الحتمية المرغوبة بشدة في الأنظمة المدمجة الحرجة، لكن الحصول على هذه الحتمية في تطبيقات المعالج الواحد يتطلب الانتقال من الجدولة الساكنة البسيطة (التي يمكن أن تؤدي للانحصار) إلى الجدولة الديناميكية الأكثر تعقيداً، وهذا يأتي مع عبء تنفيذي.

Kahn Process Networks

- **Another problem: memory overhead with buffers.**
  Potentially, it is possible that the memory need for buffers grows unlimited.

  Possible approaches:
  - For some applications and restrictions on inputs, FIFO bounds can be mathematically derived in design to avoid FIFO overflows
  - FIFO bounds can be grown on demand
  - Blocking writes can be used so that a process blocks if a FIFO is full (this deviates from the KPN semantics and may lead to deadlocks, which add further implementation issues)

- **Kahn process networks are relatively strong in their expressive power but sometimes cannot be implemented efficiently.**

Introduce more limitations so that you can get efficient implementations.

هذه الشريحة تستكمل النقاش حول التحديات العملية لتطبيق شبكات Kahn Process Networks (KPN) وتركز على مشكلة أخرى مهمة بالإضافة إلى الجدولة، وهي **استخدام الذاكرة (Memory Usage)** وتأثيرها على الكفاءة.

1. **"Another problem: memory overhead with buffers."**
   - هذه هي المشكلة الجديدة التي يتم تقديمها. في KPN ، القنوات (Channels) التي تربط بين العمليات يتم تمثيلها عادةً على أنها **مخازن مؤقتة (buffers)** من نوع FIFO في الذاكرة. عندما تنتج عملية توكن وترسله إلى قناة، يتم وضع هذا التوكن في المخزن المؤقت لتلك القناة. وعندما تستهلك عملية أخرى توكن من القناة، يتم قراءته وإزالته من المخزن المؤقت.
   - "memory overhead with buffers" تعني أن هذه المخازن المؤقتة تستهلك مساحة من الذاكرة. هذا الاستهلاك هو "عبء إضافي (overhead) " لأن مساحة الذاكرة هذه لا تُستخدم لتخزين البيانات الرئيسية للنظام أو كود العمليات نفسه، بل فقط لتسهيل الاتصال بين العمليات.
2. **"Potentially, it is possible that the memory need for buffers grows unlimited."**
   - هذه هي المشكلة المحتملة الكبرى المتعلقة بالذاكرة. في نموذج KPN النظري، غالباً ما يُفترض أن القنوات لها **سعة غير محدودة (unlimited capacity)** . هذا الافتراض يسهل التحليل النظري ويضمن أن العمليات المنتجة لن تنحصر أبداً بسبب امتلاء القناة (لنتذكر أن الكتابة غير مانعة).

- ولكن في نظام مدمج حقيقي، الذاكرة محدودة جداً. إذا كانت العملية المنتجة تنتج توكنات بشكل أسرع بكثير من العملية المستهلكة، يمكن أن يتراكم عدد كبير جداً من التوكنات في المخزن المؤقت للقناة. في أسوأ السيناريوهات (النظرية مع سعة غير محدودة)، يمكن أن يستمر المخزن المؤقت في النمو إلى ما لا نهاية، مما سيؤدي حتماً إلى استهلاك كل الذاكرة المتاحة في النظام الحقيقي (memory overflow) وتوقف النظام أو تعطله.

3. **"Possible approaches:"** هذه النقطة تقدم بعض الحلول أو الأساليب للتعامل مع مشكلة نمو المخازن المؤقتة في التنفيذات العملية لKPN :

- **" For some applications and restrictions on inputs, FIFO bounds can be mathematically derived in design to avoid FIFO overflows"**
  - في بعض الحالات، إذا كان لدينا فهم جيد لسلوك العمليات (كم تنتج وكم تستهلك في كل مرة تنفذ فيها) وقيود معينة على معدل وصول المدخلات، يمكننا **اشتقاق حدود رياضية (mathematically derived bounds)** على الحد الأقصى لعدد التوكنات التي يمكن أن تتواجد في كل قناة في أي وقت. إذا تمكنا من حساب هذه الحدود، يمكننا عندها تخصيص حجم ثابت للمخزن المؤقت في الذاكرة يساوي هذا الحد الأقصى، مما يضمن عدم حدوث تجاوز (overflow) للمخزن المؤقت. هذه الطريقة تتطلب تحليلاً دقيقاً للنظام وليست ممكنة لجميع الشبكات، خاصة المعقدة منها أو التي تعتمد على مدخلات متغيرة جداً.
- **" FIFO bounds can be grown on demand"**
  - هذه مقاربة تعتمد على تخصيص الذاكرة للمخازن المؤقتة ديناميكياً أثناء وقت التشغيل. بدلاً من تخصيص حجم ثابت وكبير مقدماً، يبدأ المخزن المؤقت بحجم صغير، وإذا امتلأ، يتم تخصيص مساحة إضافية له في الذاكرة("growing on demand") . هذه الطريقة أكثر مرونة وتستخدم الذاكرة بكفاءة أعلى في المتوسط، لكنها تتطلب نظام إدارة ذاكرة ديناميكي في النظام المدمج (وهو قد يكون معقداً ويضيف عبئاً) وما زالت لا تمنع حدوث المشكلة إذا لم تتوفر مساحة ذاكرة إضافية عند الحاجة.
- **"Blocking writes can be used so that a process blocks if a FIFO is full (this deviates from the KPN semantics and may lead to deadlocks, which add further implementation issues)"**
  - هذا حل عملي ولكنه **يخالف القواعد الأساسية لنموذج KPN النظري** .الفكرة هي تغيير سلوك الكتابة من "غير مانعة" إلى "مانعة" إذا كان المخزن المؤقت للقناة ممتلئاً. بمعنى، إذا حاولت عملية إنتاج إرسال توكن إلى قناة وكان المخزن المؤقت لتلك القناة ممتلئاً بالكامل، فإن العملية المنتجة **ستنحصر (block)** ولن تتمكن من إرسال التوكن حتى تستهلك العملية القارئة توكنات من القناة فتصبح هناك مساحة متاحة.
  - **المشكلة هنا** :هذا يكسر خاصية الكتابة غير المانعة فيKPN ، والأهم أنه يمكن أن يؤدي إلى **جمود (deadlocks)** .لنتخيل عملية A تكتب إلى قناة متجهة لعملية B، وعملية B تكتب إلى قناة متجهة لعملية A (تغذية راجعة). إذا امتلأت قناة A->B وحاولت A الكتابة فانحصرت، وفي نفس الوقت امتلأت قناة B->A وحاولت B الكتابة فانحصرت، فإن كلتا العمليتين ستنحصران وتنتظران الأخرى للأبد، وهذا جمود. هذه المشكلة لا تحدث في KPN النظري لأن الكتابة غير مانعة. استخدام الكتابة المانعة يتطلب تحليلاً وتصميماً دقيقاً جداً لتجنب مثل هذه الجمود.

4. **"Kahn process networks are relatively strong in their expressive power but sometimes cannot be implemented efficiently."**

- هذه نقطة تلخيصية مهمة. نموذج KPN قوي جداً في قدرته على نمذجة أنظمة متوازية معقدة وضمان حتمية السلوك. (Expressive Power) ولكن، كما رأينا مع تحديات الجدولة ومشكلة الذاكرة للمخازن المؤقتة، فإن تطبيق KPN بشكل **فعال** على أنظمة مدمجة محدودة الموارد يمكن أن يكون صعباً.
- **"Introduce more limitations so that you can get efficient implementations."**
  - هذا هو الاستنتاج النهائي لهذه الشريحة. بما أن تطبيق KPN النظري بالكامل صعب وغير فعال على الأنظمة المدمجة، غالباً ما يتم **تقديم المزيد من القيود (limitations)** على النموذج لجعله أكثر قابلية للتطبيق والتنفيذ بكفاءة. هذه القيود قد تكون على شكل :
    - تحديد أنواع معينة من هياكل الشبكات المسموح بها.
    - فرض حدود قصوى على حجم المخازن المؤقتة (مع التعامل مع احتمالية الامتلاء بطرق لا تؤدي لجمود).
    - استخدام نماذج تدفق بيانات أخرى ذات قيود أكبر ولكنها تضمن متطلبات الموارد (مثل الذاكرة).

ربط هذا بالروبوتات:

مشكلة الذاكرة في الأنظمة المدمجة للروبوتات حقيقية جداً. معالجة البيانات الحسية (مثل الصور أو بيانات السحابة النقطية) يمكن أن تتطلب مساحة ذاكرة كبيرة جداً. إذا صممنا نظام الروبوت الخاص بنا كشبكة KPN، يجب أن نكون واعين بحجم البيانات التي قد تتراكم في القنوات بين العمليات وأنظمة إدارة هذه المخازن المؤقتة. قد نضطر إلى اختيار خوارزميات تتطلب مخازن مؤقتة أصغر يمكن التنبؤ بحجمها، أو استخدام تقنيات نمذجة أخرى أكثر ملاءمة لموارد الذاكرة المحدودة.

هذه الشريحة تكمل الصورة الكبيرة :KPN نموذج رائع للحتمية والتوازي، لكن تطبيقه العملي يتطلب حلولاً لتحديات الجدولة والذاكرة، وقد يتطلب ذلك التضحية ببعض المرونة أو إضافة قيود للحصول على تنفيذ فعال على الأجهزة المدمجة.

Synchronous Dataflow Models

- ***Dataflow process networks*** **are a particular case of Kahn process networks.**

  A particular kind of dataflow process networks, which can be efficiently implemented, are *synchronous dataflow (SDF) networks*.

- ***Synchronous dataflow networks*** **are Kahn process networks with restriction:**
  - **At each activation (firing) a process produces and consumes a fixed number of tokens on each of its outgoing and incoming channels.**
  - **For a process to fire, it must have at least as many tokens on its input channels as it has to consume.**

- **"Dataflow process networks are a particular case of Kahn process networks." :**

إن "شبكات عمليات تدفق البيانات (Dataflow process networks) "عموماً هي حالة خاصة من شبكات عمليات كان . هذا يعني أن KPN هي نموذج أوسع وأكثر عمومية.

- **"A particular kind of dataflow process networks, which can be efficiently implemented, are synchronous dataflow (SDF) networks." :**

هذه النقطة توضح أن نوعاً محدداً من شبكات عمليات تدفق البيانات، والتي يمكن تنفيذها بكفاءة، هي شبكات تدفق البيانات المتزامنة (SDF) . هذا يضع SDF كحالة خاصة أكثر تحديداً ضمن الفئة الأوسع لشبكات تدفق البيانات التي هي بدورها حالة خاصة من KPN .

- **"Synchronous dataflow networks are Kahn process networks with restriction:" :**

هذه النقطة توضح العلاقة بين SDF و KPN بشكل أدق. هي تقول إن شبكات تدفق البيانات المتزامنة (SDF) هي في الأساس شبكات عمليات كان (KPN) **مع قيود**.

ما هي هذه القيود؟ تشرحها النقاط الفرعية التالية:

"At each activation (firing) a process produces and consumes a fixed number of tokens on each of its outgoing and incoming channels."

* هذا هو القيد الأساسي الذي يميز SDF عن KPN . فيKPN ، يمكن للعملية أن تقرأ أو تكتب عدداً متغيراً من التوكنات في كل مرة تنفذ فيها حسب منطقها الداخلي وقيمة البيانات، كما رأينا في p1 التي ترسل إلى C1 أوC2 .

* فيSDF ، كل عملية تُعرّف مسبقاً بعدد ثابت من التوكنات تستهلكه من كل قناة إدخال لديها، وعدد ثابت من التوكنات تنتجه على كل قناة إخراج لديها، وذلك في كل مرة تنطلق (تُنفذ) فيها. هذه الأعداد (تسمى معدلات الإنتاج والاستهلاك production and consumption rates) تكون ثابتة ولا تتغير. مثال: عملية في SDF قد تستهلك توكنين من القناة A وتنتج توكن واحد على القناة B في كل مرة تنطلق. الأرقام 2 و 1 ثابتة لهذه العملية.

"For a process to fire, it must have at least as many tokens on its input channels as it has to consume."

هذا يوضح شرط إطلاق العملية في SDF . لكي يُسمح لعملية في SDF بالانطلاق (أي لكي يقوم المجدول بتشغيلها)، يجب أن يتوفر لديها العدد الكافي المحدد مسبقاً من التوكنات على جميع قنوات الإدخال الخاصة بها. إذا كانت العملية تستهلك توكنين من القناة A وتوكن واحد من القناة B ، فلا يمكنها الانطلاق إلا إذا كان هناك توكنان على الأقل في A وتوكن واحد على الأقل في .B هذا يشبه شرط القراءة المانعة فيKPN ، ولكن يتم تطبيقه على عدد ثابت من التوكنات وليس مجرد توكن واحد.

- إن SDF أقل قوة تعبيرية لأنها لا تسمح بالإطلاق المشروط المعتمد على قيمة البيانات. القيود المذكورة هنا (ثبات معدلات الإنتاج والاستهلاك) هي بالضبط ما يجعل من الصعب نمذجة سلوكيات مثل توجيه البيانات بناءً على قيمتها.
- وبسبب هذه القيود وثبات معدلات الإنتاج والاستهلاك، يصبح من الممكن إجراء تحليل رياضي للنظام في وقت التصميم لتحديد ما إذا كان النظام يمكن جدولته لتجنب الانحصار وتحديد أحجام المخازن المؤقتة اللازمة، وهو ما يجعل SDF قابلاً للتنفيذ بكفاءة أعلى على الأنظمة المدمجة مقارنة بـ KPN العامة.

الخلاصة:

إن SDF هي نسخة "مقيدة" من KPN. القيد الرئيسي هو أن كل عملية في SDF تستهلك وتنتج عدداً ثابتاً ومعروفاً مسبقاً من التوكنات في كل مرة تنطلق فيها. هذا القيد يجعل SDF أقل مرونة وقوة تعبيرية من KPN (لا يمكن نمذجة السلوكيات المعتمدة على قيمة البيانات بنفس السهولة)، ولكنه في المقابل يسهل تحليل النظام وجدولته وإدارة موارده (مثل الذاكرة) بكفاءة أعلى، مما يجعلها مناسبة لبعض تطبيقات الأنظمة المدمجة.

Synchronous Dataflow Models

- **Synchronous dataflow models are less expressive than Kahn process networks:**
  - **With SDF models it is impossible to express conditional firing, where a process' firing depends on a certain condition; SDF are *static* dataflow models.**

- **For the above reduced expressiveness, however, we get two nice features of SDF models:**
  1. **Possibility to produce static schedules.**
  2. **Limited and predictable amount of needed buffer space.**

هذه الشريحة تقدم مقارنة رئيسية بين SDF وشبكات Kahn Process Networks (KPN) التي ركزنا عليها حتى الآن.

- **"Synchronous dataflow models are less expressive than Kahn process networks:"** هذه هي العبارة المقارنة الأساسية. نماذج تدفق البيانات المتزامنة **أقل قوة تعبيرية (less expressive)** من شبكات Kahn Process Networks . ماذا يعني "أقل قوة تعبيرية" في هذا السياق؟

الفقرة التالية تشرح ذلك:

- **"With SDF models it is impossible to express conditional firing, where a process' firing depends on a certain condition; SDF are static dataflow models."**
  - في نماذج SDF ، **لا يمكن التعبير عن "الإطلاق المشروط (conditional firing)** ". الإطلاق (firing) هنا يعني تنفيذ العملية في KPN ، العملية "تنطلق" (تُنفذ جزء من كودها) عندما تتوفر البيانات في القنوات التي تحاول القراءة منها. هذا الإطلاق يمكن أن يكون مشروطاً ضمنياً بتوفر البيانات.

- في SDF ، إطلاق العملية **لا يعتمد على قيمة البيانات نفسها** أو شروط معقدة تعتمد على حالة غير معروفة مسبقاً. إطلاق العملية في SDF يعتمد فقط على **توفر عدد محدد مسبقاً وثابت من التوكنات على قنوات الإدخال الخاصة** بها.
- ولذلك، يُقال عن SDF إنها نماذج تدفق بيانات **ساكنة (static)** . سلوك النظام من حيث عدد مرات إطلاق كل عملية والترتيب الدوري الذي يمكنها الانطلاق به هو أمر **محدد بالكامل وقت التصميم أو الترجمة(compile time)** ، ولا يتغير بناءً على قيم البيانات الفعلية التي تتدفق أثناء التنفيذ.

المقارنة معKPN :

في KPN ، العملية p1 قرأت القيمة من I ثم قررت إرسالها إما إلى C1 أو C2 بناءً على **قيمة البيانات نفسها** (هل هي زوجية أم فردية). هذا مثال على سلوك يعتمد على البيانات. عملية p3 أيضاً في مثالنا (إذا اتبعنا الكود الأصلي) كانت تختار القراءة إما من C3 أو C4 بناءً على حالة داخلية (sw) تتغير بناءً على التنفيذ السابق. هذه الديناميكية في KPN حيث المسار الذي تسلكه البيانات أو القناة التي تُقرأ منها يمكن أن تعتمد على قيم البيانات هي ما يجعلها "أكثر قوة تعبيرية."

في المقابل، في نموذج SDF ، تكون قواعد الإنتاج والاستهلاك للتوكنات ثابتة ومعروفة مسبقاً لكل عملية. مثلاً، عملية معينة في SDF قد تُعرف بأنها تستهلك توكنين من مدخلها A وتنتج توكن واحد على مخرجها B في كل مرة تنطلق فيها. هذه الأرقام (2 و 1) ثابتة ولا تتغير بغض النظر عن قيم التوكنات.

النتيجة العملية لكون SDF ساكنة:

لأن سلوك إطلاق العمليات في SDF محدد مسبقاً ولا يعتمد على البيانات، يصبح من الممكن **بشكل عام** اشتقاق جدول ساكن (Static Schedule) لعمليات SDF يضمن عدم الانحصار وتحديد أحجام المخازن المؤقتة (القنوات) اللازمة بشكل دقيق لتجنب تجاوز الذاكرة (buffer overflow). هذا يجعل SDF مناسبة جداً للتنفيذ الفعال على الأنظمة المدمجة محدودة الموارد، حيث يمكننا تخصيص الموارد (وقت المعالج، الذاكرة) مسبقاً بكفاءة.

ولكن، الثمن هو **فقدان القدرة على نمذجة السلوكيات التي تعتمد بشكل كبير على قيمة البيانات لاتخاذ قرارات حاسمة في مسار التدفق أو شروط التنفيذ** .النماذج الساكنة مثل SDF لا تستطيع بسهولة نمذجة أشياء مثل عبارات if/else التي توجه البيانات إلى قنوات مختلفة بناءً على القيمة، أو العمليات التي تتوقف عن الإنتاج فجأة بناءً على قيمة معينة، وما إلى ذلك.

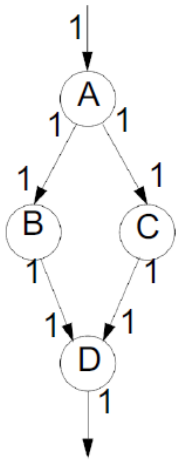
ربط هذا بالروبوتات:

- إذا كنا نصمم جزءاً من نظام الروبوت حيث تدفق البيانات وقواعد المعالجة ثابتة ومحددة (مثل سلسلة من فلاتر الإشارة الرقمية أو تحويلات هندسية معروفة)، فإن SDF قد يكون نموذجاً ممتازاً ومفصلاً للغاية يسمح لنا بتحسين استخدام الموارد على المعالج المدمج.
- ولكن إذا كان لدينا أجزاء من النظام يتغير فيها سلوك المعالجة أو مسار البيانات بشكل كبير بناءً على المدخلات الحسية (مثل وحدة التعرف على الأشياء التي قد ترسل إحداثيات الكائن عبر قناة إذا تم التعرف عليه، أو لا ترسل شيئاً إذا لم يتم العثور على شيء)،

فإن SDF قد لا تكون كافية لنمذجة هذا السلوك الديناميكي المعتمد على البيانات بشكل كامل، وقد نحتاج إلى العودة إلى نماذج أكثر قوة تعبيرية مثل KPN (مع تحديات الجدولة والذاكرة التي ناقشناها) أو نماذج أخرى هجينة.

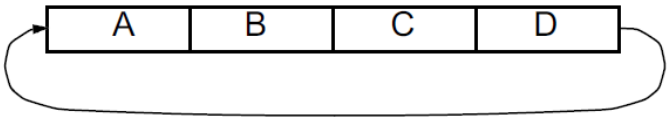
باختصار، SDF نموذج مفيد جداً وقابل للتنفيذ بكفاءة على الأنظمة المدمجة، ولكنه يفرض قيوداً على أنواع السلوكيات التي يمكن نمذجتها (لا إطلاق مشروط يعتمد على قيمة البيانات)، مما يجعله أقل قوة تعبيرية من KPN ولكنه أسهل في الجدولة وإدارة الموارد.

Synchronous Dataflow Models



- **Arcs are marked with the number of tokens produced or consumed.**
- **This is a simple "single-rate" system: every process is activated one single time before the system returns to its initial state.**

Possible static schedule:



في أي نظام روبوتي معقد، لدينا مهام متعددة تعمل في نفس الوقت تقريبًا: قراءة حساسات، معالجة صور، حساب مسارات الحركة، إصدار أوامر للمحركات، وغيرها الكثير. هذه المهام تحتاج إلى التفاعل وتبادل المعلومات فيما بينها. كيف ننظم هذا التفاعل بطريقة موثوقة وفعالة؟ هنا يأتي دور نماذج تدفق البيانات.

ماذا يعني "تدفق البيانات المتزامن"؟ ببساطة شديدة، هو طريقة لوصف النظام على أنه مجموعة من العمليات التي تتواصل مع بعضها البعض عن طريق تمرير "رموز (Tokens)" من البيانات عبر قنوات (الأقواس في الرسم). كلمة "متزامن" هنا تعني أن كل عملية **تستهلك وتنتج عددًا ثابتًا ومعروفًا من الرموز في كل مرة يتم تشغيلها**. هذا الثبات والمعرفة المسبقة هو مفتاح هذا النموذج وقوته.

بالنظر إلى الرسم البياني على اليسار:

- لدينا هنا أربع عمليات (أو مهام): A, B, C, D قد تكون وظائف برمجية أو حتى وحدات هاردوير متخصصة.
- الأسهم بينها تمثل القنوات التي تتدفق البيانات عبرها.
- الأرقام "1" المكتوبة بجانب الأسهم هي **عدد الرموز التي يتم إنتاجها أو استهلاكها** عند تشغيل العملية المتصلة بالسهم .

- فمثلاً، السهم من A إلى B وعليه الرقم 1 يعني أن العملية A عندما تعمل، تنتج 1 رمز تضعه في القناة المتجهة نحو B . والعملية B عندما تعمل، تستهلك 1 رمز من القناة القادمة من A
- نلاحظ أن كل الأقواس هنا عليها الرقم 1، سواء كانت عملية إنتاج (السهم خارج من العملية) أو استهلاك (السهم داخل إلى العملية).

النقطة الثانية الهامة ."This is a simple "single-rate" system" : بما أن كل العمليات هنا تنتج وتستهلك رمزًا واحدًا فقط في كل مرة تعمل فيها، هذا يجعل النظام "أحادي المعدل (Single-Rate) ". يعني ببساطة أننا يمكننا تشغيل كل عملية مرة واحدة (أو عدد ثابت من المرات) في دورة كاملة للنظام، وبعد هذه الدورة، يعود النظام إلى حالة "مستقرة" وجاهز لبدء الدورة التالية.

لماذا هذا مهم في الروبوتات؟ لنتخيل أن A هو مهمة قراءة بيانات حساس المسافة، B مهمة حساب المسافة الحقيقية، C مهمة تحديد ما إذا كان هناك عائق، و D مهمة إصدار أمر التوقف للمحركات إذا كان العائق قريبًا. في نظام أحادي المعدل كهذا، قراءة واحدة من الحساس تؤدي إلى حساب واحد للمسافة، وتحليل واحد للعائق، وإصدار أمر واحد للمحركات (إذا لزم الأمر). كل خطوة تحدث مرة واحدة في الدورة.

الجزء الأخير يتحدث عن :"Possible static schedule" بما أننا نعرف بالضبط كم رمزًا يتم إنتاجه واستهلاكه عند كل تشغيل، يمكننا **قبل حتى تشغيل النظام فعليًا** تحديد ترتيب ثابت لتشغيل العمليات يضمن أن كل عملية تجد البيانات التي تحتاجها (الرموز التي تستهلكها) وتضع البيانات التي تنتجها في مكانها الصحيح للعملية التالية. هذا الترتيب الثابت يسمى "الجدول الساكن(Static Schedule) ".

في هذا المثال البسيط، الجدول الساكن الموضح هو :A **ثم B ثم C ثم D ، ثم نعود لتكرار التسلسل (ممثلاً بالدارة العائدة من D إلى A)**
لماذا هذا الترتيب منطقي هنا؟

1. **A تعمل أولاً :**لأنها لا تستهلك أي رموز من عمليات أخرى (ليس هناك أسهم داخلة إليها)، وهي تنتج رموزًا تحتاجها B و C
2. **Bو C تعملان بعد A :**لأنهما تستهلكان رموزًا تنتجها A . يمكن تشغيلهما بأي ترتيب(B ثم C أو C ثم B)أو حتى بالتوازي إذا كان المعالج يدعم ذلك، طالما أن A قد انتهت ووفرت الرموز لهما. الرسم هنا يوضح تسلسل A ثم B ثم C
3. **D تعمل بعد B و C :**لأنها تستهلك رموزًا تنتجها B و C. لا يمكن لـ D أن تعمل قبل أن تنتهي B و C من عملهما وتوفير الرموز اللازمة.

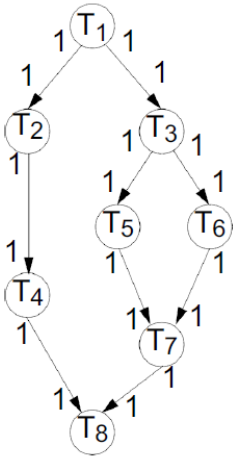
لماذا الجداول الساكنة مفيدة جدًا في الأنظمة المدمجة والروبوتات؟

- **التنبؤية (Predictability) :**نعرف بالضبط متى ستعمل كل مهمة، مما يسهل التحقق من الالتزام بالمواعيد الزمنية (Real-time constraints).هذا حيوي للروبوتات التي تحتاج للاستجابة بسرعة وبشكل متوقع لبيئتها.
- **كفاءة استخدام الموارد (Resource Efficiency) :** يمكن تخصيص الذاكرة (للقنوات التي تحمل الرموز) ووقت المعالج بشكل ثابت ومعروف مسبقًا، مما يقلل الحمل الزائد (Overhead) ويجعل النظام أكثر كفاءة.
- **تجنب الجمود (Deadlock Prevention) :**إذا تم إيجاد جدول ساكن صالح، فهذا يضمن أن النظام لن يصل إلى حالة جمود حيث تنتظر العمليات بعضها البعض إلى ما لا نهاية.
- **بساطة التنفيذ (Simplicity of Implementation) :** بمجرد تحديد الجدول، يمكن تنفيذه كحلقة تكرار بسيطة تشغل العمليات بالترتيب المحدد.

إذاً، في هذا السلايد ، تعرفنا على أساسيات نماذج تدفق البيانات المتزامنة، وكيف أن ثبات معدلات الإنتاج والاستهلاك يسمح لنا ببناء أنظمة أحادية المعدل وتحديد جداول تشغيل ثابتة وموثوقة، وهو حجر زاوية في تصميم الأنظمة المدمجة الفعالة والقابلة للتنبؤ بها لروبوتاتنا الذكية.

Synchronous Dataflow Models

Our example from Lecture 1:



A static schedule:

T1	T2	T4	T3	T5	T6	T7	T8

ما زلنا في إطار الأنظمة التي نعرف فيها بالضبط كمية البيانات (الرموز) التي تتبادلها العمليات. وهذا المثال هو فقط لتوضيح كيف يمكن تطبيق نفس الفكرة على شبكة أكبر من المهام المترابطة.

لدينا الآن 8 عمليات أو مهام، مرمزة من T1 إلى T8 ، كل سهم عليه الرقم "1"، مما يؤكد أن هذا لا يزال نظام "أحادي المعدل (Single-Rate) " . كل مهمة هنا، عندما يتم تشغيلها مرة واحدة في الدورة الكاملة، ستستهلك رمزًا واحدًا من كل قناة تأتي إليها وتنتج رمزًا واحدًا لكل قناة تخرج منها.

سوف نحلل هذا الرسم من منظور الاعتماديات (Dependencies) .

بالنظر إلى الجدول الساكن الموضح على اليمين :هو تسلسل معين لتشغيل هذه المهام **T1, T2, T4, T3, T5, T6, T7, T8**

لماذا هذا التسلسل هو جدول ساكن صالح؟ لنتبع تدفق البيانات ونرى:

لقد نجحنا في إيجاد ترتيب لتشغيل كل المهام مرة واحدة في تسلسل يضمن أن كل مهمة يتم تشغيلها فقط بعد توفر جميع البيانات التي تحتاجها من المهام التي تسبقها. هذا التسلسل يمثل دورة عمل كاملة واحدة للنظام، يعود بعدها إلى نقطة البداية (نهاية T8 تعني اكتمال الدورة).

كيف نربط هذا بمثال روبوتي أكثر تعقيدًا؟ لنتخيل أن هذا جزء من نظام رؤية روبوتي:

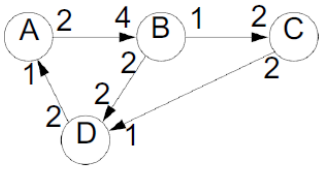
- **T1 :**مهمة الحصول على إطار جديد من الكاميرا.

- **T2**: مهمة معالجة سريعة لهذا الإطار (مثل تصحيح الألوان).
- **T3**: مهمة معالجة أخرى على الإطار (مثل البحث عن علامات معينة).
- **T4**: استخلاص الميزات من الإطار المعالج بواسطة T2
- **T5**: التعرف على الأجسام بناءً على علامات وجدت بواسطة T3.
- **T6** : تتبع الأجسام التي تم التعرف عليها من الإطار السابق وبيانات T3 الحالية.
- **T7**: دمج (Fusion) كل هذه المعلومات: الميزات، التعرف، والتتبع، لتحديد موقع وحالة الأجسام المهمة في المشهد.
- **T8**: استخدام هذه المعلومات النهائية لتحديث نموذج البيئة الخاص بالروبوت أو إصدار أمر للمحركات للتفاعل مع جسم معين.

كما نرى، كل مهمة تنتج قطعة بيانات (رمز) تستهلكها مهمة أو أكثر لاحقًا في pipeline المعالجة. النظام أحادي المعدل يعني أن معالجة إطار كاميرا واحد تؤدي في النهاية إلى تحديث واحد لنموذج البيئة/إصدار أمر واحد للمحركات.

العثور على جدول ساكن صالح مثل **T1, T2, T4, T3, T5, T6, T7, T8**(أو أي ترتيب صالح آخر) هو جزء أساسي من عملية تصميم النظام. هذا الجدول يضمن أن كل خطوة في معالجة إطار الكاميرا تحدث في الترتيب الصحيح، بدون انتظار غير ضروري (بمجرد توفر البيانات)، وبشكل متكرر وقابل للتنبؤ به في كل دورة. هذا هو جوهر قوة نماذج تدفق البيانات المتزامنة في الأنظمة التي تتطلب دقة وتوقيتًا، مثل الروبوتات.

Deriving a static schedule for SDF



- **For a correct synchronous dataflow network there exists a sequence of firings which returns the network in its original state.**

  This sequence represents a static schedule which has to be repeated in a cycle.

- **The schedule is such that a finite amount of memory is required (no infinite buffers)**

Problem

How to derive such a cyclic schedule?

الآن سوف نتناول **كيفية اشتقاق جدول ساكن لأنظمة تدفق البيانات المتزامنة** . في الفقرات السابقة، فهمنا ما هو نموذج SDF وما هو الجدول الساكن، ورأينا أمثلة بسيطة. الآن، السؤال هو: إذا كان لدينا شبكة معقدة من العمليات، كيف نجد هذا الجدول الساكن؟

بالنظر إلى الرسم البياني الجديد على اليسار. أول شيء سنلاحظه هو أن الأرقام على الأسهم لم تعد كلها "1". هذا يعني أننا انتقلنا من نظام "أحادي المعدل (Single-Rate) " إلى نظام **"متعدد المعدلات (Multi-Rate) "** في هذه الأنظمة، قد تحتاج عملية معينة إلى العمل عدة مرات مقابل مرة واحدة تعملها عملية أخرى في الدورة الكاملة للنظام.

بملاحظة الأرقام بجانب الأسهم:

- السهم من A إلى B : بجانب A الرقم 2، وبجانب B الرقم 4. هذا يعني أنه في كل مرة تعمل فيها العملية A ، تنتج رمزين (Tokens) تضعهما في القناة المتجهة إلى B . وفي كل مرة تعمل فيها العملية B ، تستهلك 4 رموز من هذه القناة.
- وهكذا لبقية الأسهم... كل سهم له معدل إنتاج عند مصدره ومعدل استهلاك عند وجهته.

النقاط الهامة التي يذكرها السلايد:

1. **"For a correct synchronous dataflow network there exists a sequence of firings which returns the network in its original state. This sequence represents a static schedule which has to be repeated in a cycle."** هذه نقطة أساسية. ليس كل شبكة تدفق بيانات متزامنة "صحيحة" أو "قابلة للجدولة (Schedulable) " الشبكة الصحيحة هي التي يمكن إيجاد تسلسل لتشغيل عملياتها (جدول ساكن) بحيث يعود النظام في نهاية هذا التسلسل إلى نفس الحالة التي بدأ منها (أي أن كميات الرموز في جميع القنوات تعود إلى ما كانت عليه). هذا التسلسل هو دورتنا التشغيلية التي تتكرر.
2. **"The schedule is such that a finite amount of memory is required (no infinite buffers)"**. هذا هو أحد أكبر المزايا لنماذج SDF الصحيحة. بما أننا نعود إلى الحالة الأصلية في نهاية كل دورة، فإن عدد الرموز الأقصى الذي يمكن أن يتواجد في أي قناة في أي وقت خلال الدورة يكون محدودًا ومعروفًا. لا نحتاج إلى مساحة تخزين (Buffers) لا نهائية للرموز، وهذا حيوي جدًا في الأنظمة المدمجة ذات الموارد المحدودة مثل الروبوتات.

"Problem: How to derive such a cyclic schedule?" :كيف نجد هذا التسلسل ؟

الخطوة الأولى والأكثر أهمية هي تحديد **"متجه التكرار (Repetition Vector)** "هذا المتجه، لنسميه q، يحتوي على عدد المرات التي يجب أن يتم فيها تشغيل كل عملية (A, B, C, D...) في دورة واحدة كاملة للنظام لضمان التوازن.

لكي يعود النظام لحالته الأصلية، يجب أن يكون إجمالي عدد الرموز التي تنتجها عملية X للقناة المتجهة نحو Y مساويًا لإجمالي عدد الرموز التي تستهلكها Y من نفس القناة، ولكن **خلال دورة كاملة واحدة** تشمل تشغيل كل عملية العدد المحدد في متجه التكرار.

إذا كانت العملية X تعمل بمقدار qX مرة في الدورة، وتنتج PXY رمزًا في كل مرة تعمل فيها على القناة المتجهة نحوY ، فإن إجمالي ما تنتجه هو qX×PXY. وإذا كانت العملية Y تعمل بمقدار qY مرة في الدورة، وتستهلك CXY رمزًا في كل مرة تعمل فيها من القناة القادمة منX ، فإن إجمالي ما تستهلكه هو qY×CXY

لضمان التوازن في الدورة الكاملة، يجب أن يكون qX×PXY=qY×CXY :وذلك لكل قناة (سهم) في الشبكة.

من خلال هذا، نحصل على نظام من المعادلات الخطية. حل هذا النظام يعطينا القيم النسبية لـ qA,qB,qC,qD . نحن نبحث عن أصغر حل صحيح وموجب (غير صفري). هذا الحل هو متجه التكرار الذي يخبرنا، مثلاً، أننا نحتاج لتشغيل A مرتين، و Bمرة واحدة، و C أربع مرات، وهكذا، في كل دورة كاملة للنظام.

كمثال على الرسم لدينا (مع افتراض أن الأرقام بجانب المصدر هي معدل الإنتاج وبجانب الوجهة هي معدل الاستهلاك):

- A -> B: qA×2=qB×4
- B -> C: qB×1=qC×2
- B -> D: qB×2=qD×2
- C -> D: qC×2=qD×1
- D -> A: qD×2=qA×1

بحل هذا النظام نحصل على قيم متجه التكرار [qA,qB,qC,qD]

الخطوة الثانية :بمجرد معرفة متجه التكرار، تبدأ مهمة **إيجاد التسلسل الفعلي للتشغيل (الجدول الساكن)** .هذا التسلسل يجب أن يراعي شرطين رئيسيين:

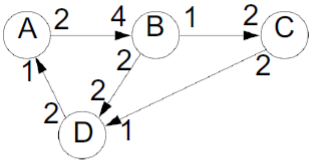
1. يجب أن يتم تشغيل كل عملية A بعدد مرات يساوي qA و كل عملية B بعدد مرات يساوي qB وهكذا، خلال التسلسل بأكمله.
2. لا يمكن تشغيل عملية ما إلا إذا توفر لديها العدد الكافي من الرموز على جميع قنوات الدخل الخاصة بها، حسب معدلات الاستهلاك المحددة.

البحث عن هذا التسلسل يتطلب خوارزميات خاصة مع الأخذ في الاعتبار عدد مرات التكرار لكل عقدة وكميات الرموز في القنوات (مستويات الـ buffer) .الهدف هو بناء تسلسل من تشغيل العمليات يحقق شروط الاعتمادية وعدد مرات التكرار المطلوب لكل عملية في متجه q، ويضمن عدم حدوث جمود (Deadlock) وأن النظام يعود لنفس حالة الرموز في النهاية.

لماذا هذا مهم جدًا في هندسة الروبوتات؟ لأن الروبوت يحتاج إلى التصرف بشكل يمكن التنبؤ به. معرفة جدول التشغيل الثابت مسبقًا يعني أننا نعرف بالضبط متى سيتم قراءة الحساسات، متى ستتم المعالجة، ومتى سيتم إصدار الأوامر للمحركات، كل ذلك ضمن دورة زمنية محددة. هذا يساعد في ضمان الاستجابة في الوقت المناسب (Real-time Response) ويسهل تحليل أداء النظام والتأكد من صحته.

إذاً، هذا السلايد يطرح السؤال: كيف ننتقل من مجرد رسم لشبكة SDF إلى جدول تشغيل عملي؟ الإجابة تبدأ بحساب متجه التكرار بناءً على معادلات توازن الرموز، ثم البحث عن تسلسل صالح لتشغيل العمليات يحقق متطلبات التكرار والاعتماديات.

Deriving a static schedule for SDF



- **Along the periodic sequence of firing, on each arc the same number of tokens has to be produced and consumed.**

  a, b, c, d: the number of firings, during a period, for process A, B, C, D.

Balance equations:

$$2a - 4b = 0$$
$$b - 2c = 0$$
$$2c - d = 0$$
$$2b - 2d = 0$$
$$2d - a = 0$$

$$\begin{bmatrix} 2 & -4 & 0 & 0 \\ 0 & 1 & -2 & 0 \\ 0 & 0 & 2 & -1 \\ 0 & 2 & 0 & -2 \\ -1 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$$

كيفية إيجاد عدد مرات تشغيل كل عملية في الدورة الواحدة:

- **"Along the periodic sequence of firing, on each arc the same number of tokens has to be produced and consumed."**

هذا هو المبدأ الأساسي للحفاظ على حالة مستقرة ومتكررة للنظام. إجمالي الرموز التي تخرج من طرف السهم يجب أن يساوي إجمالي الرموز التي يتم استهلاكها من الطرف الآخر، ولكن **فقط عند النظر لدورة كاملة واحدة** من تشغيل النظام وفقًا للجدول الساكن.

السلايد يقدم لنا الرموز **a, b, c, d**: لتمثل عدد مرات تشغيل (Firings) العمليات A, B, C, D على التوالي، خلال فترة (دورة) واحدة. هذه الرموز هي مكونات "متجه التكرار" الذي تحدثنا عنه سابقًا. هدفنا هو إيجاد أصغر قيم صحيحة وموجبة (ليست كلها أصفار) لهذه الرموز.

"Balance equations:": هذه هي المعادلات التي تعبر عن مبدأ توازن الرموز لكل سهم في الشبكة. كيف نحصل عليها؟ لكل سهم يخرج من عملية X ويدخل عملية Y بمعدل إنتاج PXY عند X ومعدل استهلاك CXY عند Y ، فإن المعادلة هي: (عدد مرات تشغيل X) × (معدل إنتاج X على هذا السهم) = (عدد مرات تشغيل Y) × (معدل استهلاك Y من هذا السهم) qX×PXY=qY×CXY (أو بإعادة الترتيب لتكون المعادلة مساوية للصفر: qX×PXY−qY×CXY=0

الآن لنطبق هذا على المعادلات المعروضة (مع استخدام الرموز a, b, c, d بدلاً من qA,qB,qC,qD):

1. **2a - 4b = 0**: هذه المعادلة تتعلق بالسهم من A إلى B . الرسم يظهر معدل إنتاج 2 عند A ومعدل استهلاك 4 عند B على هذا السهم.

2. **$b - 2c = 0$**: هذه المعادلة تتعلق بالسهم من B إلى C . الرسم يظهر معدل إنتاج 1 عند B ومعدل استهلاك 2 عند C على هذا السهم.

3. **$2c - d = 0$**: هذه المعادلة تتعلق بالسهم من C إلى D . الرسم يظهر معدل إنتاج 2 عند C ومعدل استهلاك 1 عند D على هذا السهم.

4. **$2b - 2d = 0$** : هذه المعادلة تتعلق بالسهم من B إلى D . الرسم يظهر معدل إنتاج 2 عند B ومعدل استهلاك 2 عند D على هذا السهم.

5. **$-a + 2d = 0$**: هذه المعادلة تتعلق بالاتصال بين A و D . الرسم يظهر معدل إنتاج 2 عند D واستهلاك 1 عند A

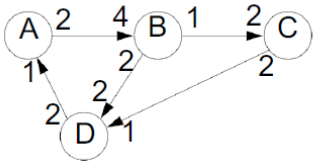
الآن، لدينا نظام من 5 معادلات خطية في 4 متغيرات(a, b, c, d) ، وهذا النظام متجانس (كل المعادلات تساوي صفر). هدفنا هو إيجاد حل غير صفري لهذا النظام يمثل أصغر قيم صحيحة وموجبة لـ a, b, c, d.

يمكننا حل هذا النظام فينتج متجه التكرار هو [a,b,c,d]=[4,2,1,2]

ماذا يعني هذا في سياق نظام الروبوت؟ يعني أنه في كل دورة كاملة من تشغيل هذا النظام المدمج (ربما دورة معالجة واحدة لإطار حساس، أو دورة تحكم واحدة)، يجب أن يتم تشغيل العملية A عدد 4 مرات، والعملية B عدد 2 مرة، والعملية C عدد 1 مرة، والعملية D عدد 2 مرة.

هذه القيم (4, 2, 1, 2) هي عدد مرات "إطلاق (Firing) "كل عملية في الدورة. الخطوة التالية هي استخدام هذه الأرقام لبناء التسلسل الفعلي للتشغيل (الجدول الساكن)، مع مراعاة توفر الرموز اللازمة لكل عملية قبل تشغيلها.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

$$\Gamma q = 0$$

- Γ: topology matrix of the graph
- q: firing vector
- 0: vector of zeros

- **If there is no $q \neq 0$ which satisfies the equation above $\Rightarrow$ there is no static schedule (there is a *rate inconsistency* between processes).**

Balance equations:

$$2a - 4b = 0$$
$$b - 2c = 0$$
$$2c - d = 0$$
$$2b - 2d = 0$$
$$2d - a = 0$$

$$\begin{bmatrix} 2 & -4 & 0 & 0 \\ 0 & 1 & -2 & 0 \\ 0 & 0 & 2 & -1 \\ 0 & 2 & 0 & -2 \\ -1 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$$

السلايد يقدم لنا الأداة الرياضية الرسمية للتحقق مما إذا كان نظام تدفق البيانات المتزامن لدينا "صحيحًا" وقابلاً للجدولة الساكنة أم لا

"If there is no q≠0 which satisfies the equation above => there is no static schedule (there is a rate inconsistency between processes)."

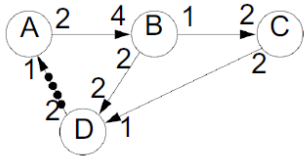
هذه الجملة هي قلب الموضوع. إذا قمنا بتحليل شبكة SDF الخاصة بنا، وبنينا مصفوفة الطوبولوجيا Γ الخاصة بها، وحاولنا حل المعادلة $\Gamma q=0$، ووجدنا أن الحل الوحيد الممكن هو $q=0$ (متجه الأصفار)، فهذا يعني للأسف أن شبكتا **غير قابلة للجدولة الساكنة** في صورتها الحالية.

ماذا يعني هذا من الناحية العملية في نظام روبوتي؟ هذا يعني أن هناك **عدم اتساق في المعدلات (Rate Inconsistency)** بين العمليات. كأن تقول مثلاً: العملية A تنتج البيانات بسرعة أكبر بكثير مما يمكن للعملية B استهلاكه، وفي نفس الوقت العملية C تحتاج بيانات من B بمعدل عالٍ جدًا. هذه المعدلات المتضاربة تجعل من المستحيل إيجاد جدول تشغيل متكرر يعيد النظام لحالته الأصلية دون أن تمتلئ قنوات بيانات معينة إلى ما لا نهاية، أو أن تنتظر عمليات أخرى إلى الأبد بيانات لن تصل أبدًا بالكمية المطلوبة في دورة زمنية محدودة ومتكررة.

في عالم الروبوتات، اكتشاف عدم الاتساق هذا مبكرًا أمر بالغ الأهمية. لا يمكننا بناء نظام تحكم روبوتي موثوق ودقيق بتصميم فيه عدم اتساق جوهري في تدفق البيانات بين مهامه. إذا وجدنا أن المعادلة $\Gamma q=0$ لا تملك حلاً غير صفري لمتجه q، فهذا يعني أن تصميمنا الأساسي (من حيث تعريف المهام ومعدلات تبادل البيانات بينها) يحتاج إلى مراجعة وتعديل. قد نحتاج إلى تغيير خوارزميات المعالجة لتقليل متطلبات الاستهلاك، أو تغيير معدلات أخذ العينات من الحساسات، أو إعادة هيكلة بعض المهام لضمان التوازن.

إذن، المعادلة $\Gamma q=0$ ليست مجرد مسألة رياضية، بل هي اختبار حاسم لـ "صحة" تصميم شبكة SDF من حيث قابلية الجدولة الساكنة باستخدام ذاكرة محدودة. إيجاد حل غير صفري لـ q يخبرنا أن الجدول الساكن موجود (وكم مرة يجب أن تعمل كل عملية)، والفشل في إيجاد حل غير صفري يخبرنا أن هناك مشكلة أساسية في تصميم تدفق البيانات يجب إصلاحها.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

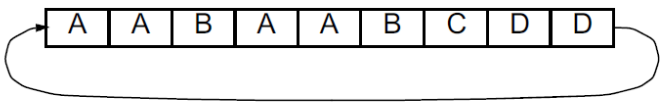
$$\Gamma q = 0$$

- **Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).**

  For our SDF graph, this solution is: a=4, b=2, c=1, d=2.

  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

الآن سننتقل من المعادلات النظرية إلى تنفيذ الجدول الساكن.

لقد رأينا الرسم البياني، وتعلمنا أن المعادلة $\Gamma q=0$ هي مفتاح إيجاد عدد مرات تشغيل كل عملية في الدورة الواحدة (متجه التكرار q).

نحن نبحث عن **أصغر متجه حل صحيح وموجب** للمعادلة $\Gamma q=0$.هذا هو الحل غير الصفري الذي يمثل الحد الأدنى لعدد مرات تشغيل كل عملية لتحقيق توازن الرموز في دورة كاملة.

وكما حسبنا سابقًا بناءً على المعادلات المقدمة، فإن الحل لهذا الرسم هو:

- **a = 4** (العملية A تعمل 4 مرات في الدورة)
-) **b = 2** (العملية B تعمل مرتين في الدورة)
-) **c = 1** (العملية C تعمل مرة واحدة في الدورة)
-) **d = 2** (العملية D تعمل مرتين في الدورة)

هذا المتجه [4,2,1,2]هو متجه الإطلاق (Firing Vector) أو متجه التكرار (Repetition Vector) الذي يخبرنا "كم مرة" يجب أن يتم إطلاق كل مهمة خلال دورة واحدة كاملة للنظام.

"A possible schedule:" :هذا هو مثال على التسلسل الفعلي لتشغيل هذه المهام في دورة واحدة**A, A, B, A, A, B, C, D, D** :

- ظهرت A 4 مرات.
- ظهرت B مرتين.
- ظهرت C مرة واحدة.
- ظهرت D مرتين

.إجمالي عدد مرات الإطلاق في هذا التسلسل هو 4 + 2 + 1 + 2 = 9 إطلاقات. هذا التسلسل المكون من 9 خطوات يمثل دورة واحدة كاملة للنظام. بعد تنفيذ هذا التسلسل مرة واحدة، يعود النظام إلى حالته الأصلية من حيث عدد الرموز في القنوات، ويكون جاهزًا لتكرار نفس التسلسل للدورة التالية.

هناك ملاحظة مهمة جدًا أسفل الجدول "The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D -> A." :

لنتذكر أن الجدول الساكن يجب أن يحترم الاعتماديات: لا يمكن تشغيل عملية إلا إذا توفر لديها العدد الكافي من الرموز التي تحتاجها من القنوات المتجهة إليها. في الشبكات التي تحتوي على حلقات تغذية راجعة(Feedback loops) ، مثل السهم الذي يعود من D إلى A في رسمنا، قد تحتاج بعض العمليات التي تقع في بداية الحلقة إلى رموز تأتي من عمليات تقع لاحقًا في الحلقة.

في هذه الحالة بالذات، العملية A تحتاج رموزًا من D . إذا بدأنا النظام بـ0 رمز في القناة من D إلىA ، فإن A قد لا تستطيع العمل في بداية الجدول إذا كانت تحتاج رموزًا لم تنتجها D بعد. ولكن الجدول يظهر أن A تعمل 4 مرات في البداية قبل أن تعمل D

هنا يأتي دور **الرموز الأولية (Initial Tokens)** . هذه الرموز هي كمية البيانات التي يجب أن تكون موجودة في بعض القنوات **عندما يبدأ النظام عمله للمرة الأولى** .هذه الرموز الأولية لا تنتجها أي عملية في الدورة الأولى، بل هي أشبه بـ"تمهيد" للقنوات للسماح للجدول بالبدء والعمل بسلاسة بدون جمود.

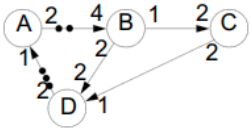
في مثالنا، القناة من D إلى A تحتاج إلى أن تبدأ بـ 4 رموز. هذه الرموز الأربعة تسمح للعملية A بالعمل 4 مرات في بداية الدورة حسب ما يقتضيه الجدول (حتى لو كانت A تستهلك رمزًا واحدًا فقط في كل مرة من هذه القناة حسب الرسم). بعد أن تستهلك A رموزها الأولية، ستقوم D بإنتاج الرموز في وقت لاحق من الجدول، مما يحافظ على التوازن ويضمن أن القناة تعود إلى حالتها الأولية (4 رموز) في نهاية الدورة لتكون جاهزة للدورة التالية.

لماذا هذا مهم جدًا في هندسة الروبوتات؟

- **التشغيل الأول :**أنظمة الروبوتات لا تبدأ من الفراغ دائمًا. قد تحتاج بيانات أولية (مثل قراءة أولية للحساسات، أو تقدير أولي للحالة) لتمكين مهام المعالجة والتحكم من البدء. الرموز الأولية في نموذج SDF تمثل هذه البيانات الأولية.
- **التحكم في التوقيت :**معرفة عدد الرموز الأولية المطلوبة أمر حاسم لضمان أن الجدول الساكن ممكن التنفيذ على أرض الواقع في النظام المدمج. إذا لم توفر العدد الكافي من الرموز الأولية، فقد يتوقف النظام (يحدث جمود) في بداية التشغيل لأنه ينتظر بيانات لن تأتيه.
- **التصميم والتحليل :**خوارزميات إيجاد الجداول الساكنة لأنظمة SDF لا تجد الجدول فقط، بل يمكنها أيضًا حساب الحد الأدنى لعدد الرموز الأولية المطلوبة في كل قناة لجعل الجدول ممكنًا. هذا جزء من عملية التحقق من صحة التصميم وقابليته للتنفيذ.

إذاً، بمجرد إيجاد متجه التكرار (عدد مرات تشغيل كل عملية)، يمكن لخوارزميات بناء الجدول الساكن إيجاد تسلسل فعلي للتشغيل (مثل A, A, B, A, A, B, C, D, D) وتحديد ما إذا كانت هناك حاجة لرموز أولية في قنوات معينة لضمان عدم حدوث جمود وتمكين تنفيذ الجدول بشكل دوري.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

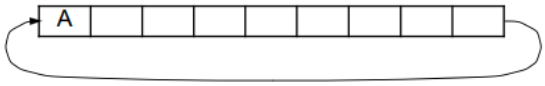
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

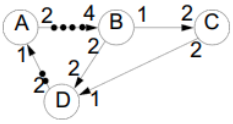
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

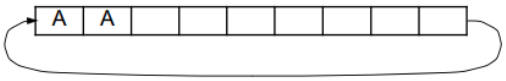
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

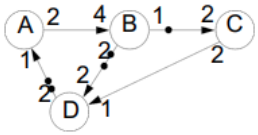
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

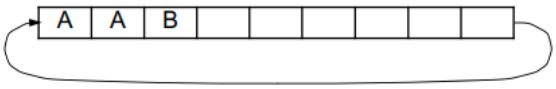
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

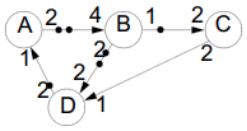
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

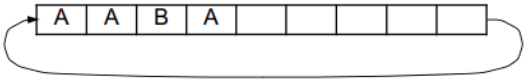
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

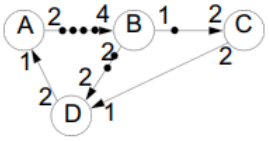
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

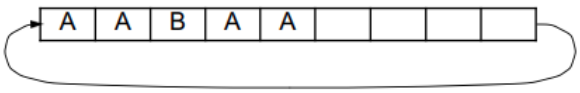
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

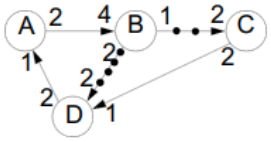
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

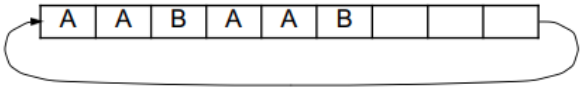
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

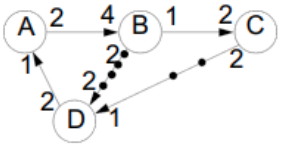
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

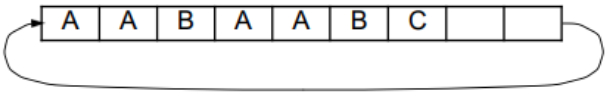
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

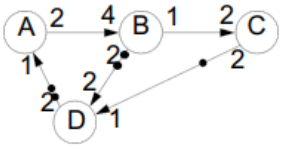
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

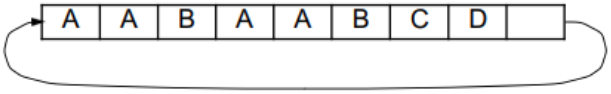
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

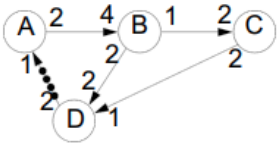
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

Deriving a static schedule for SDF



For a given SDF network (graph) we get equation:

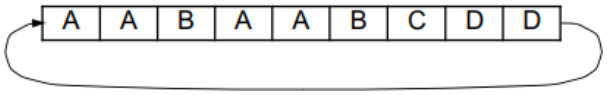
$$\Gamma q = 0$$

- Among possible solutions for vector *q*, we are interested in the smallest positive integer vector (smallest sum of the elements).

  For our SDF graph, this solution is:
  a=4, b=2, c=1, d=2.

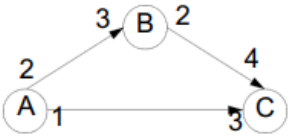
  a, b, c, d indicate how often each task is activated during one period.

A possible schedule:



The schedule is possible, without deadlock, only if 4 initial tokens are provided on the channel D → A.

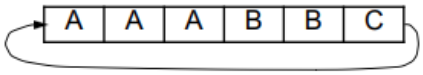
Deriving a static schedule for SDF



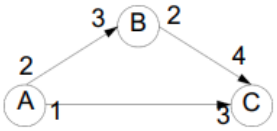
$$\begin{bmatrix} 2 & -3 & 0 \\ 0 & 2 & -4 \\ 1 & 0 & -3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0$$

Solution: a=3, b=2, c=1.

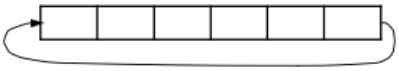
Possible schedule:



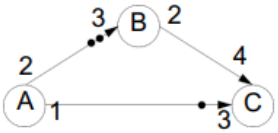
Deriving a static schedule for SDF



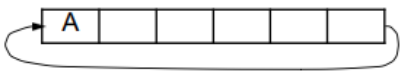
AB	0
BC	0
AC	0



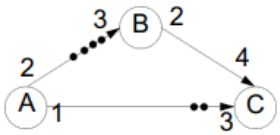
Deriving a static schedule for SDF



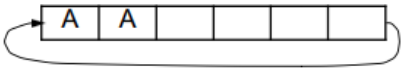
		A
AB	0	2
BC	0	0
AC	0	1



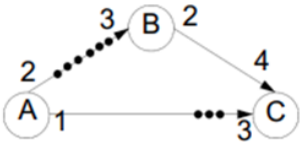
Deriving a static schedule for SDF



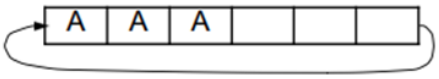
	A	A				
AB	0	2	4			
BC	0	0	0			
AC	0	1	2			



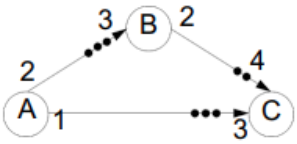
Deriving a static schedule for SDF



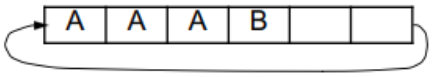
	A	A	A			
AB	0	2	4	6		
BC	0	0	0	0		
AC	0	1	2	3		



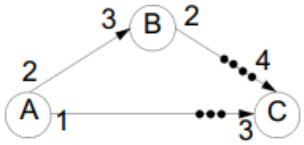
Deriving a static schedule for SDF



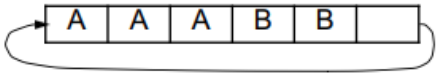
	A	A	A	B		
AB	0	2	4	6	3	
BC	0	0	0	0	2	
AC	0	1	2	3	3	



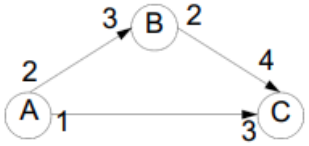
Deriving a static schedule for SDF



	A	A	A	B	B	
AB	0	2	4	6	3	0
$\overline{BC}$	0	0	0	0	2	4
$\overline{AC}$	0	1	2	3	3	3



Deriving a static schedule for SDF



	A	A	A	B	B	C
AB	0	2	4	6	3	0	0
$\overline{BC}$	0	0	0	0	2	4	0
$\overline{AC}$	0	1	2	3	3	3	0