Lecture 11: HW3, Rest of Parallel Patterns, Load Balancing

G63.2011.002/G22.2945.001 · November 16, 2010

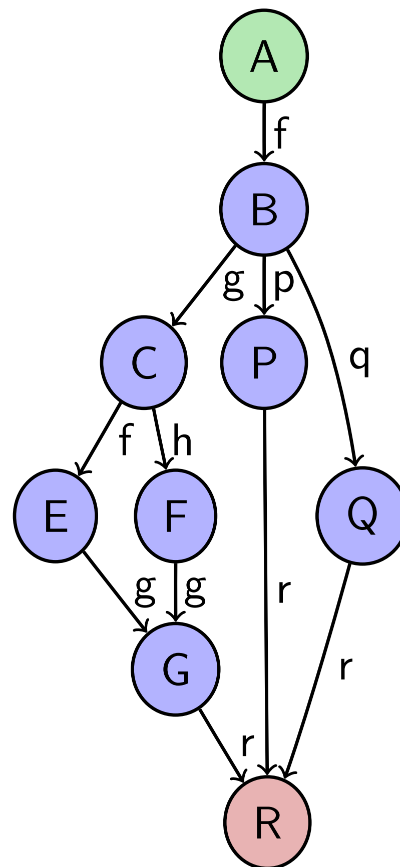
Outline

Divide-and-Conquer

General Data Dependencies

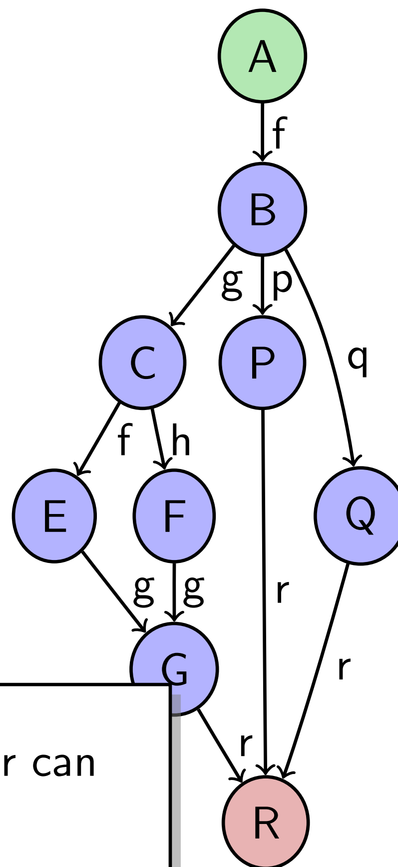
General Dependency Graphs

$B = f(A)$
$C = g(B)$
$E = f(C)$
$F = h(C)$
$G = g(E,F)$
$P = p(B)$
$Q = q(B)$
$R = r(G,P,Q)$



General Dependency Graphs

B = f(A)
C = g(B)
E = f(C)
F = h(C)
G = g(E,F)
P = p(B)
Q = q(B)
R = r(G,P,Q)



Great: All patterns discussed so far can be reduced to this one.

Cilk

```
cilk int fib (int n)
{
  if (n < 2) return n;
  else
  {
    int x, y;

    x = spawn fib (n-1);
    y = spawn fib (n-2);

    sync;

    return (x+y);
  }
}
```

Features:

- Adds keywords `spawn`, `sync`, (`inlet`, `abort`)
- Remove keywords → valid (seq.) C

Timeline:

- Developed at MIT, starting in '94
- Commercialized in '06
- Bought by Intel in '09
- Available in the Intel Compilers

Cilk

```
cilk int fib (int n)
{
  if (n < 2) return n;
  else
  {
    int x, y;

    x = spawn fib (n-1);
    y = spawn fib (n-2);

    sync;

    return (x+y);
  }
```

Efficient implementation?

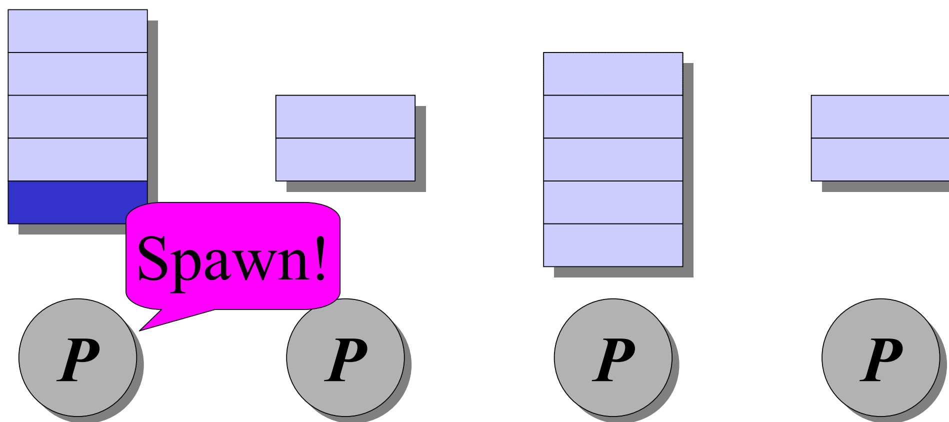
Features:

- Adds keywords `spawn`, `sync`, (`inlet`, `abort`)
- Remove keywords → valid (seq.) C

Timeline:

- Developed at MIT, starting in '94
- Commercialized in '06
- Bought by Intel in '09
- Available in the Intel Compilers

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.

With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.

With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.

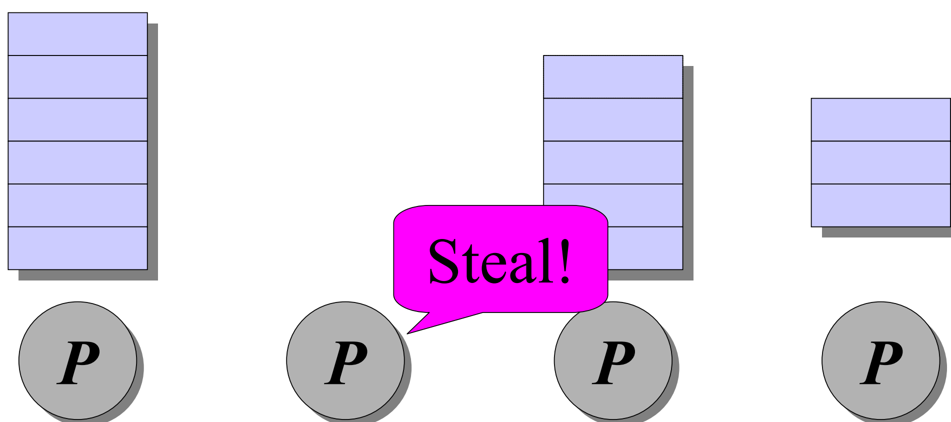
Return!

P P P P

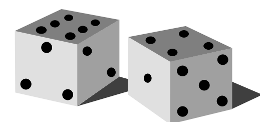
With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



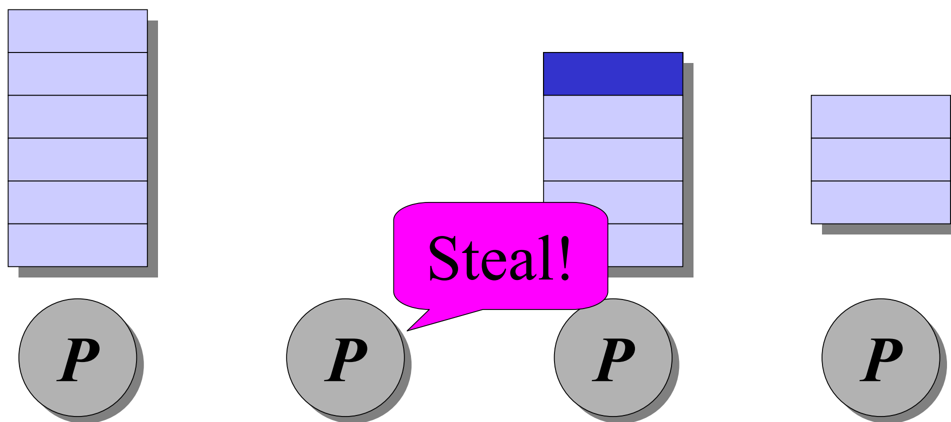
When a processor runs out of work, it ***steals*** a thread from the top of a ***random*** victim's deque.



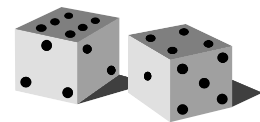
With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



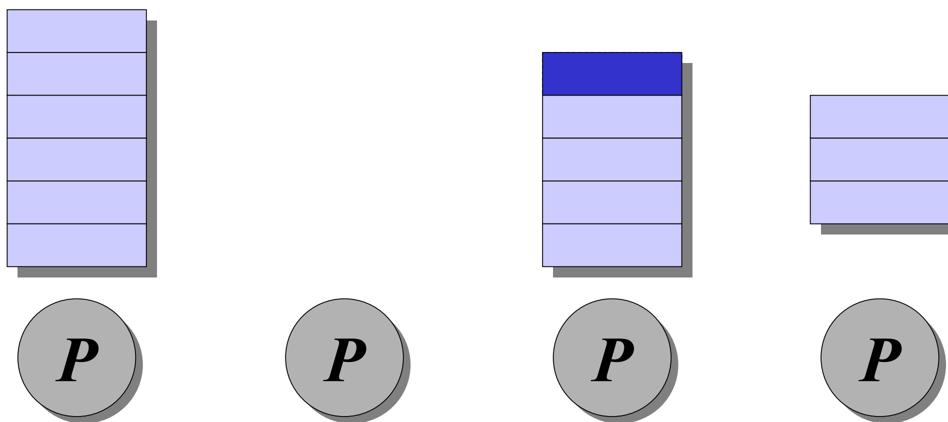
When a processor runs out of work, it ***steals*** a thread from the top of a ***random*** victim's deque.



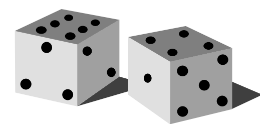
With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



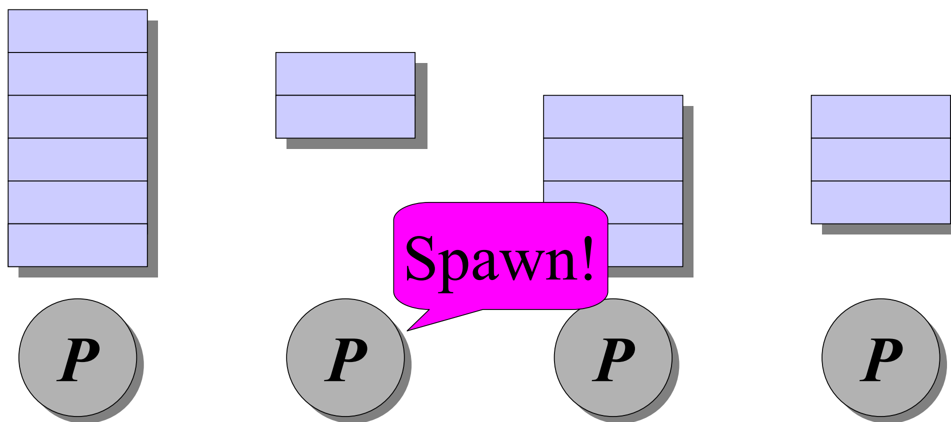
When a processor runs out of work, it ***steals*** a thread from the top of a ***random*** victim's deque.



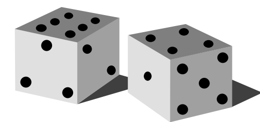
With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



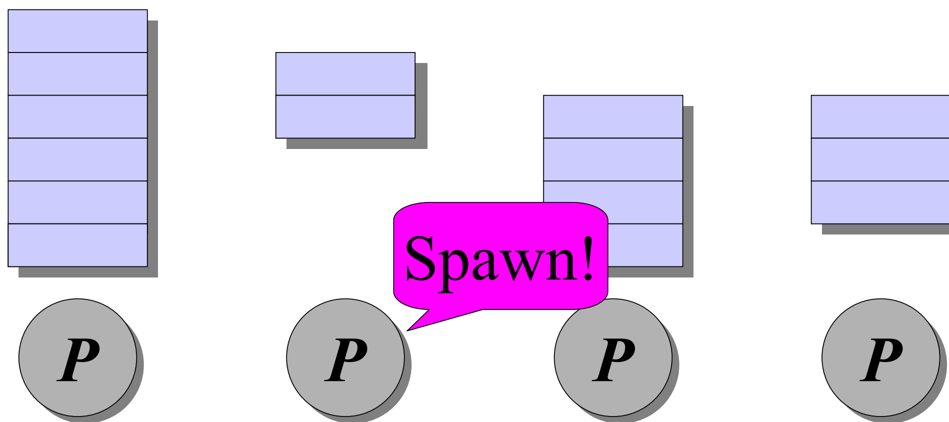
When a processor runs out of work, it ***steals*** a thread from the top of a ***random*** victim's deque.



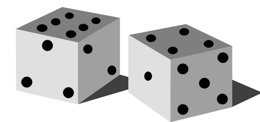
With material by
Charles E. Leiserson (MIT)

Work-Stealing

Each processor maintains a ***work deque*** of ready threads, and it manipulates the bottom of the deque like a stack.



When a processor runs out of ... rom the ... s deque.



Why is Work-Stealing better than a Task Queue?

With material by
Charles E. Leiserson (MIT)

General Graphs: Issues

- Model can accommodate 'speculative execution'
  - Launch many different 'approaches'
  - Abort the others as soon as one satisfactory one emerges.
- Discover dependencies, make up schedule at run-time
  - Usually less efficient than the case of known dependencies
  - Map-Reduce absorbs many cases that would otherwise be general
- On-line scheduling: complicated
- Not a good fit if a more specific pattern applies
- Good if inputs/outputs/functions are (somewhat) heavy-weight

Questions?

?