Lecture 23

Scheduling Parallel Computations onto Processors

work + span

work + span predict

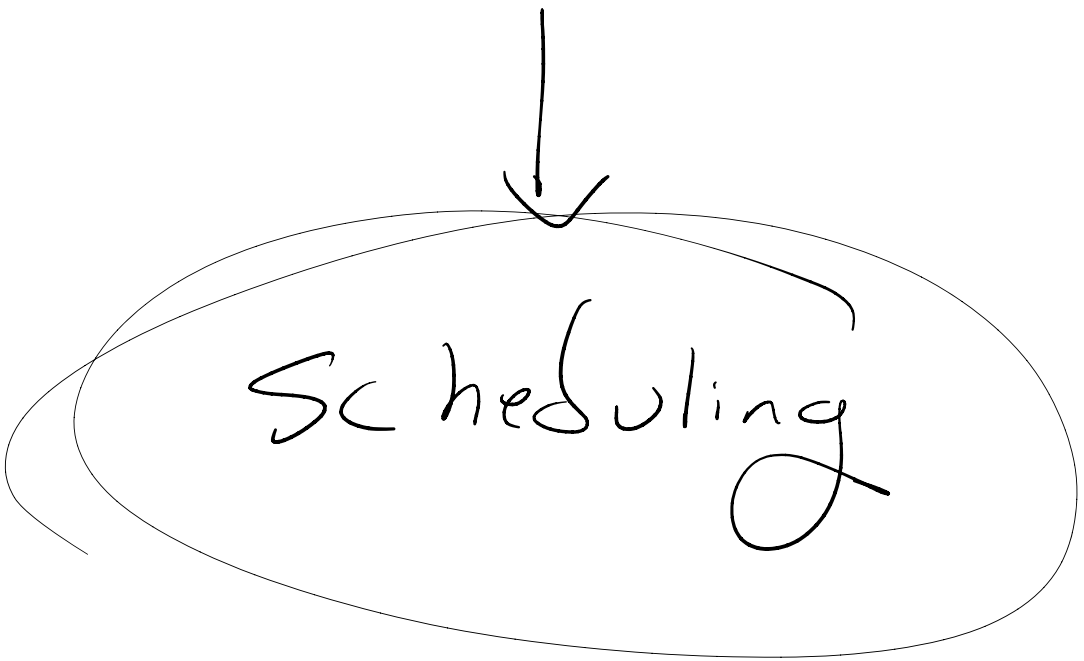
running time on

fixed # p

of processors

Brent's principle

Program can be run in time $O(\max(\frac{W}{P}, S))$



Idea

$W = 100$

$S = 50$

$P = 2$

$\frac{W}{P} = 50 = S$

time is ≈ 50

$W = 100$

$S = 50$

$P = 1$

$\frac{W}{P} = 100$

$S = 50$

not enough proc.

time ≈ 100

$W = 100$

$S = 50$

$P = 3$

$\frac{W}{P} = 33.\overline{3}$

$S = 50$

too many procs

time ≈ 50

SML/NJ doesn't do parallel scheduling

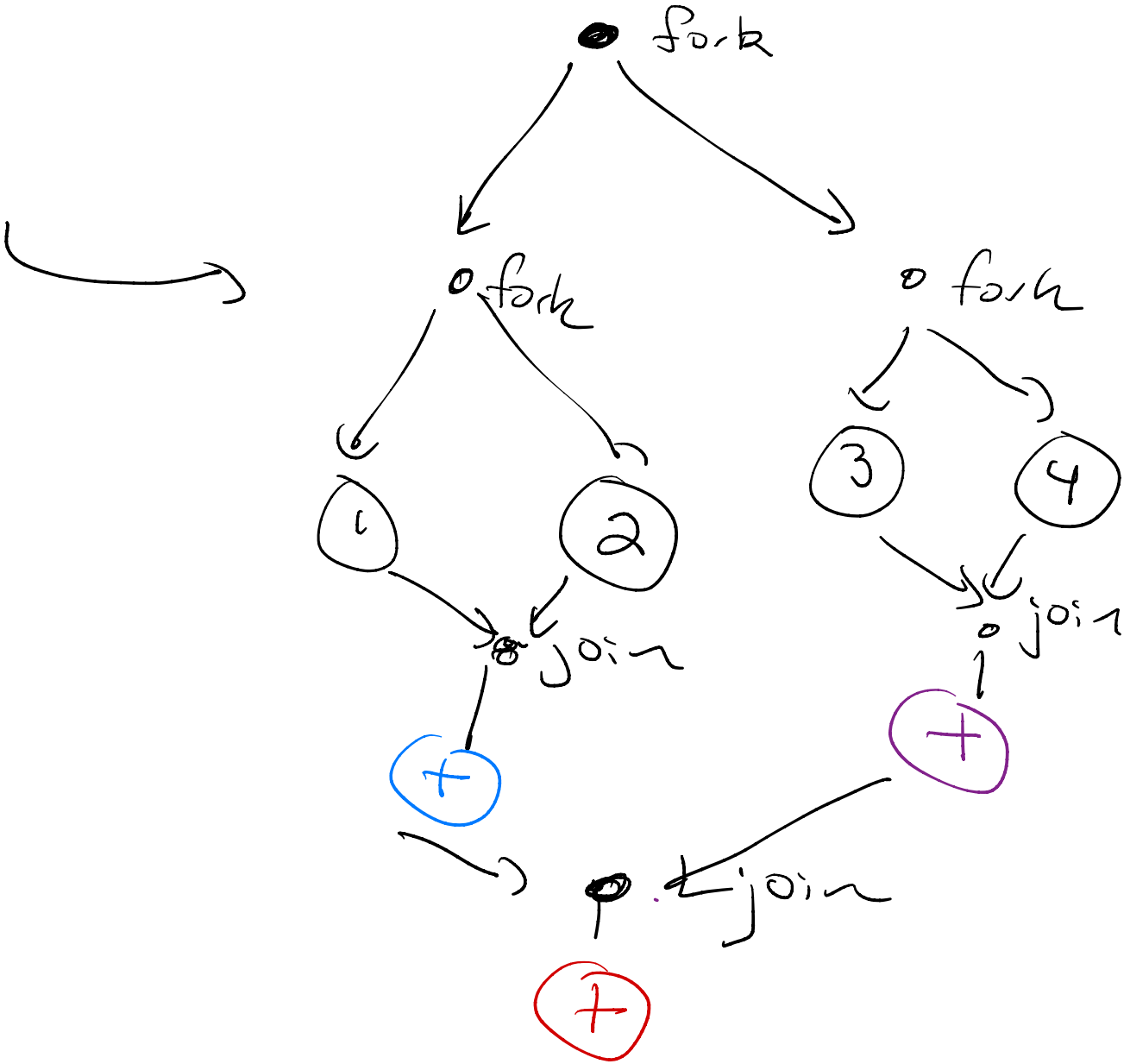
Manticore
MultiMLTon
NESL
Haskell

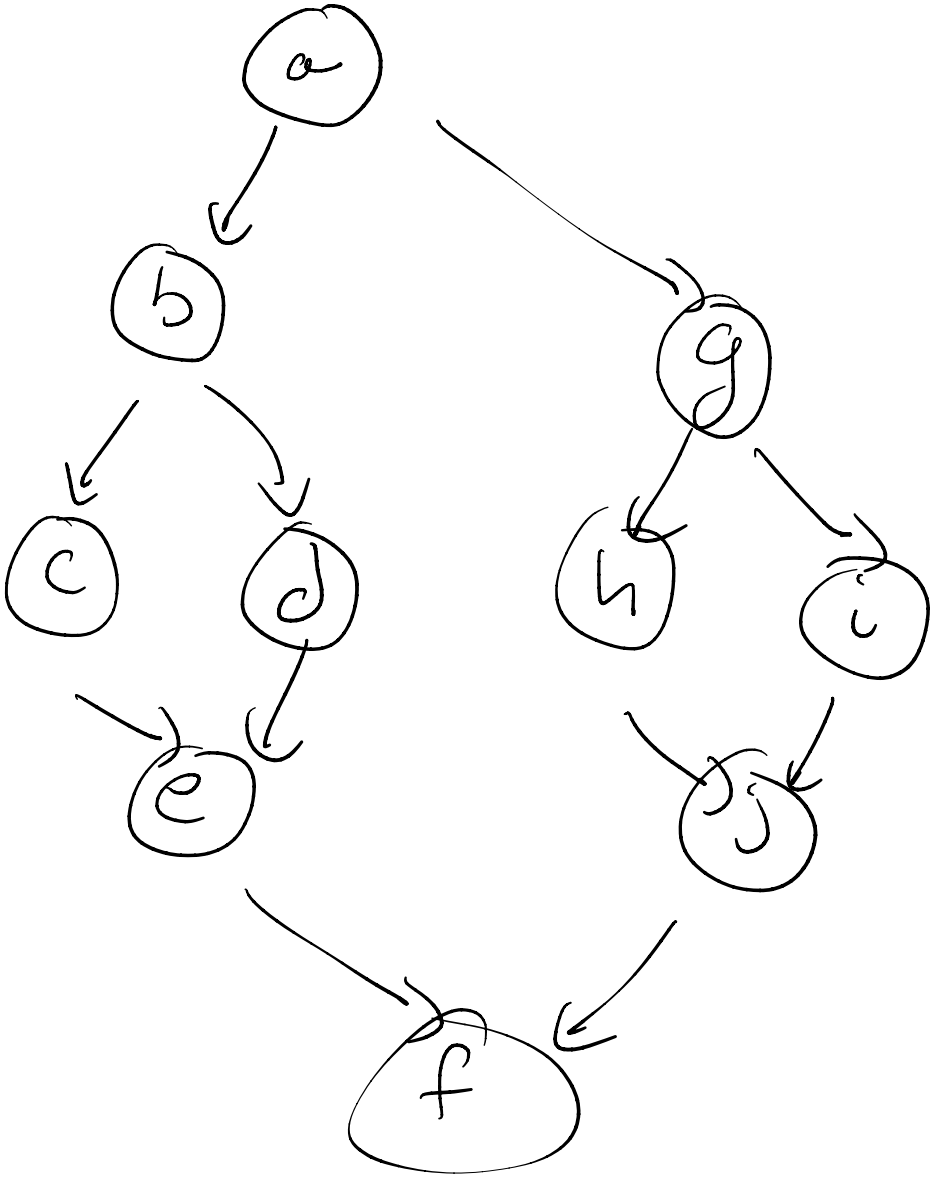
research!

Problem:
overhead

Cost graph → incrementally computed

(1 + 2) + (3 + 4)





directed acyclic graph — DAG

↓

dependency graph

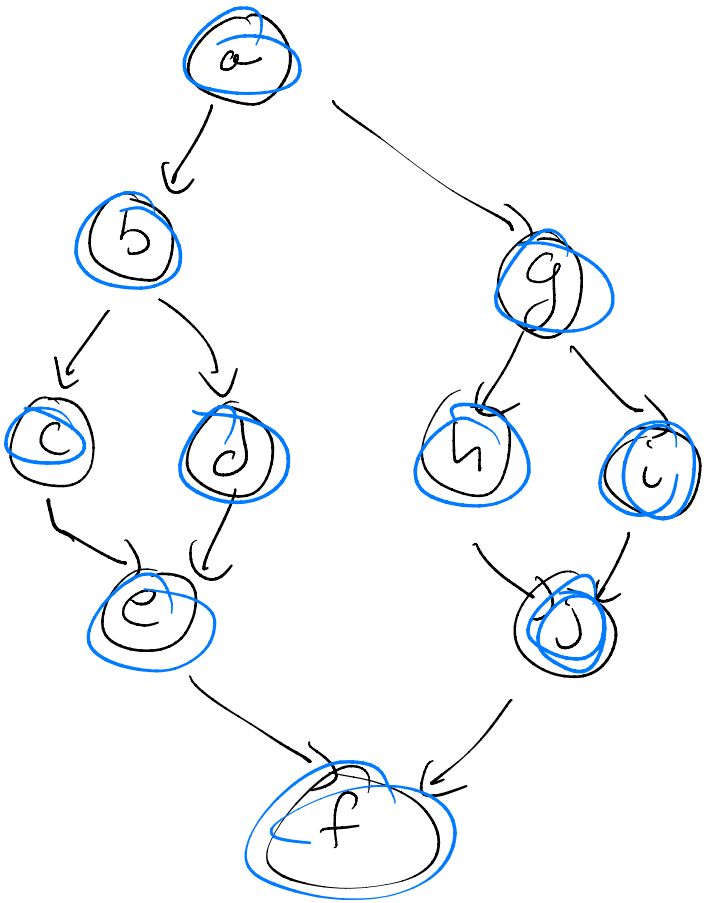
work: # of nodes

span: length of the biggest path from top to bottom

Schedule: each processor does a node at each step

Rules: each processor can do at most 1 thing per step

only do a node when the dependencies (parents) are done

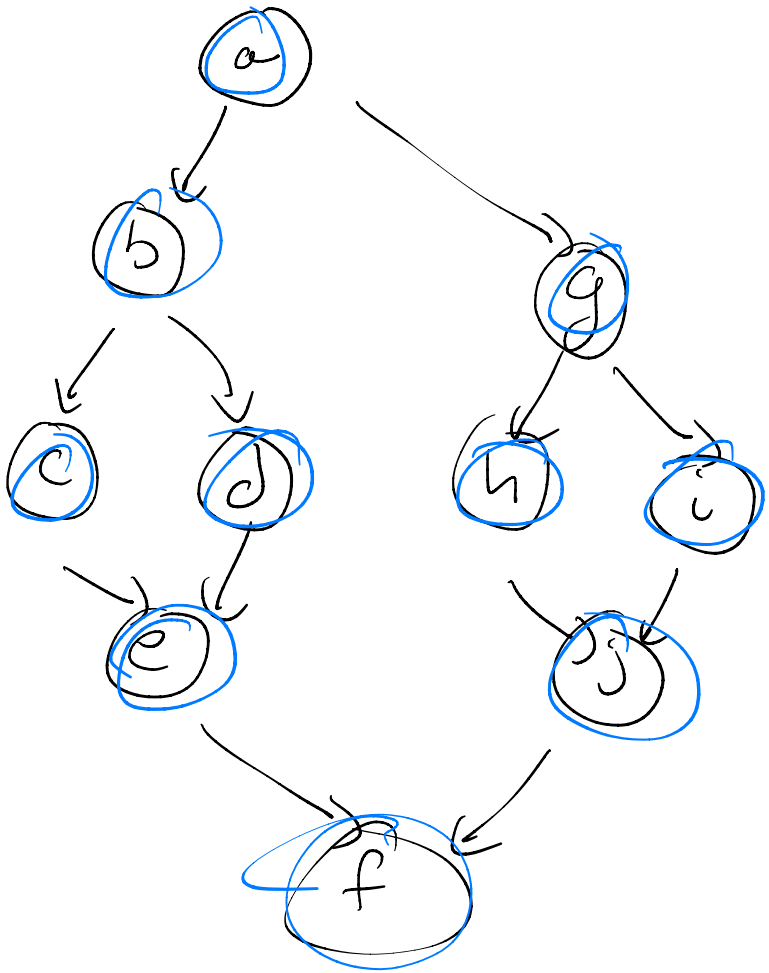


DFS

depth first scheduling

$p = 2$

Step	P1	P2
1	a	idle
2	b	g
3	c	d
4	e	h
5	i	
6	j	
7	f	
...		



BFS

breadth first

$p = 2$

Step	P1	P2
1	a	
2	b	g
3	c	d
4	h	i
5	e	j
6	f	
...		

Issues:

- not really 1 step: fork/join
  expensive
- locality
  memory access
  (cache/RAM/HD/network)
- communication

"Effects"

What can a program of a type do?

pure { - return a value of that type

computational effects
- infinite loop
- raise an exception
- print to the screen (testing)
- reading text input (controller)

new type: unit

values ()

ops let val () = e in e' end

$T_1 * T_2$ pairs

$T_1 * T_2 * T_3$ triples

unit tuple of 0 things

```
fun f(x: int): unit =
    case x of
        0 => ()
      | _ => raise Fail ""
```

2 behaviors

print : string → unit

print "hello" : unit

↦ ()

prints to the screen

TextIO.inputLine TextIO.stdIn

: string option

- asks the user for input
- if it gets something, then is that string