CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

FROM

MODERN ALGORITHMS WORKSHOP

Parallel Algorithms

Prof. Charles E. Leiserson

Dr. Tao B. Schardl

September 19, 2018

Outline

- Introduction
- Cilk Model
- Detecting Nondeterminism
- What Is Parallelism?
- Scheduling Theory Primer
- *Lunch Break*
- Analysis of Parallel Loops
- Case Study: Matrix Multiplication
- Case Study: Jaccard Similarity
- Post-Moore Software

What Is Parallelism?

Execution Model

```
int fib (int n) {
  if (n < 2) return n;
  else {
    int x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return x + y;
  }
}
```

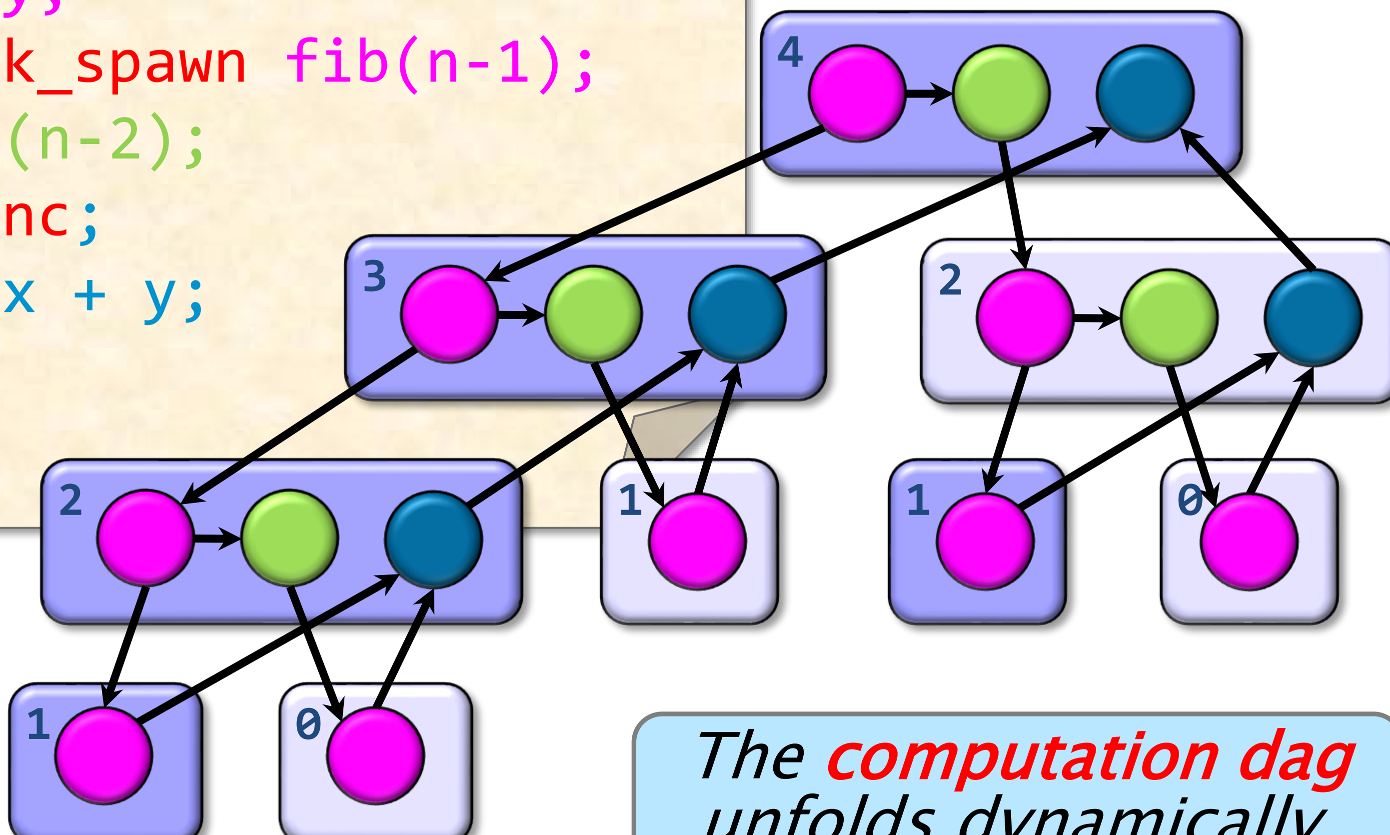
Example:

`fib(4)`

Execution Model

```
int fib (int n) {
  if (n < 2) return n;
  else {
    int x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return x + y;
  }
}
```

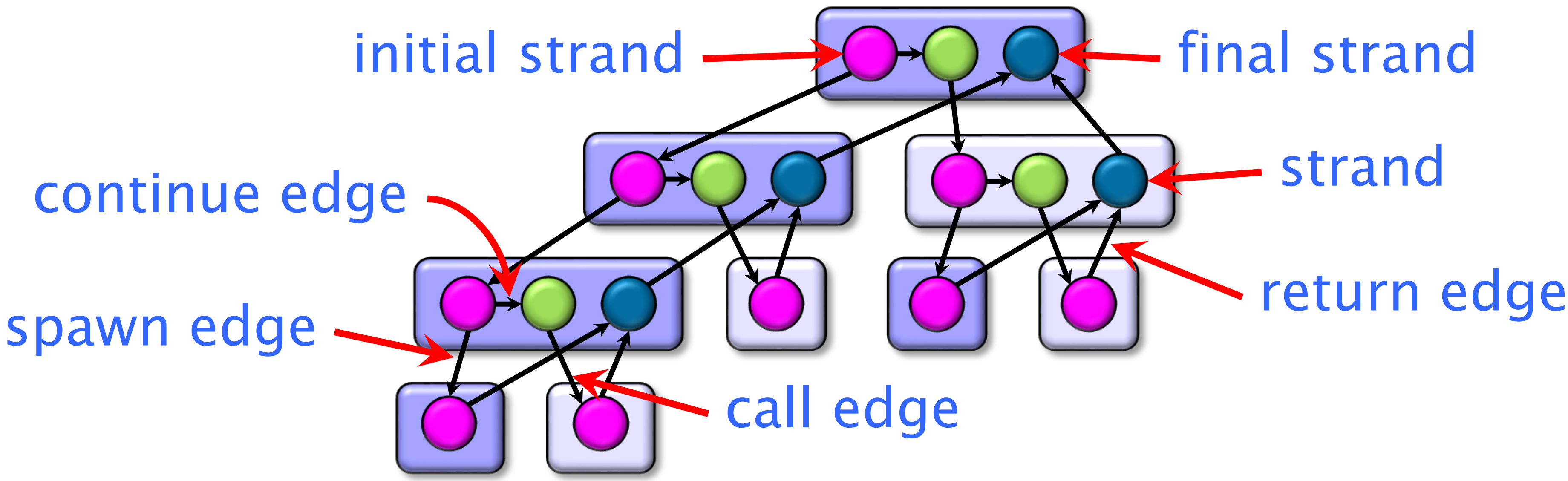
Example:
`fib(4)`



"Processor oblivious"

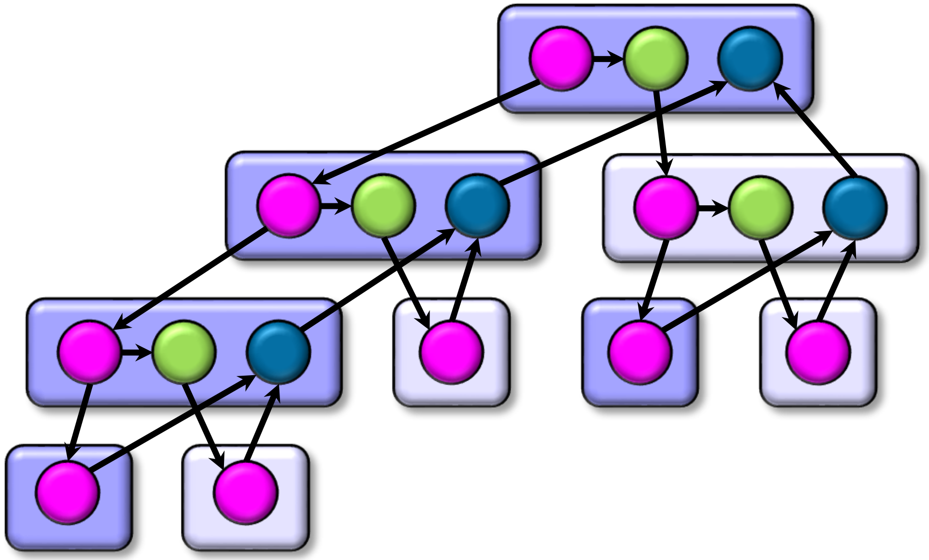
The ***computation dag*** *unfolds dynamically.*

Computation Dag



- A parallel instruction stream is a dag G = (V, E).
- Each vertex v ∈ V is a strand: a sequence of instructions not containing a spawn, sync, or return from a spawn.
- An edge e ∈ E is a spawn, call, return, or continue edge.
- Loop parallelism (`cilk_for`) is converted to spawns and syncs using recursive divide-and-conquer.

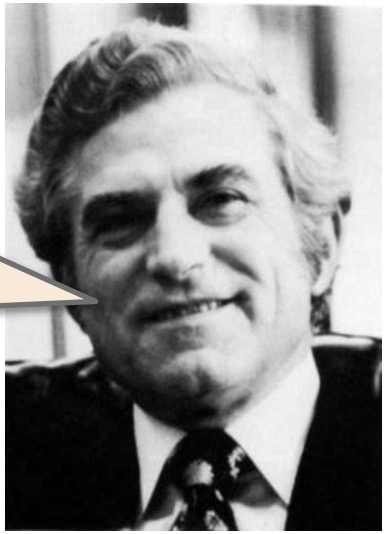
How Much Parallelism?



Assuming that each strand executes in unit time, what is the parallelism of this computation?

Amdahl's "Law"

If 50% of your application is parallel and 50% is serial, you can't get more than a factor of 2 speedup, no matter how many processors it runs on.

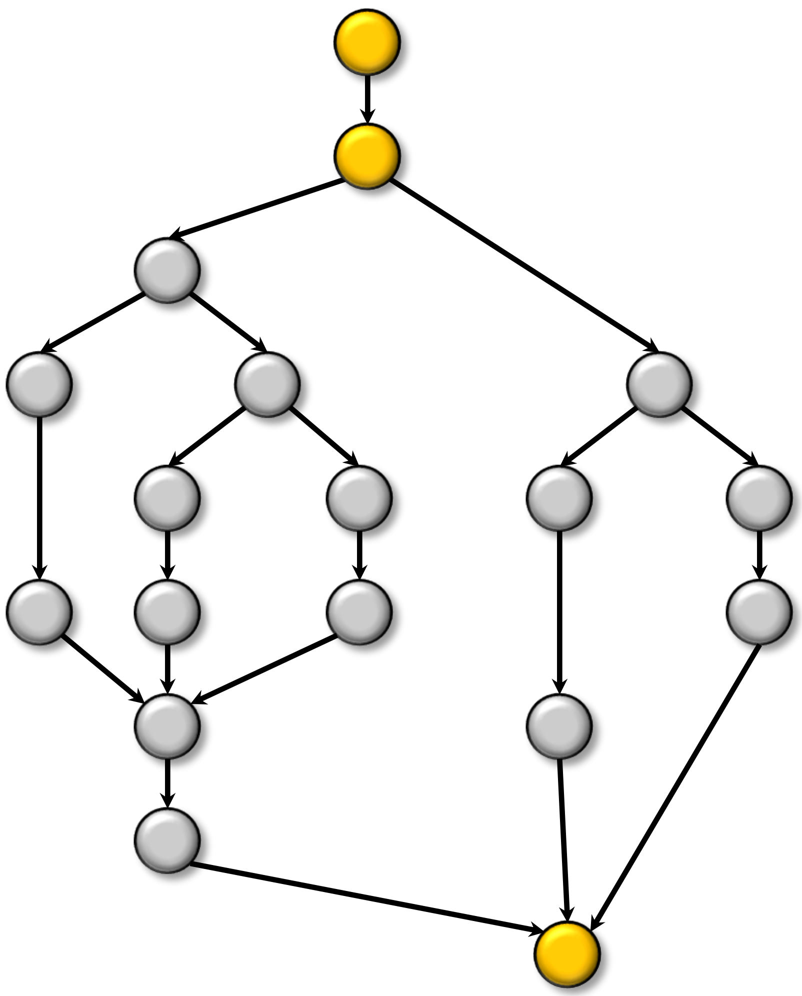


Gene M. Amdahl

In general, if a fraction α of an application must be run serially, the speedup can be at most $1/\alpha$.

Quantifying Parallelism

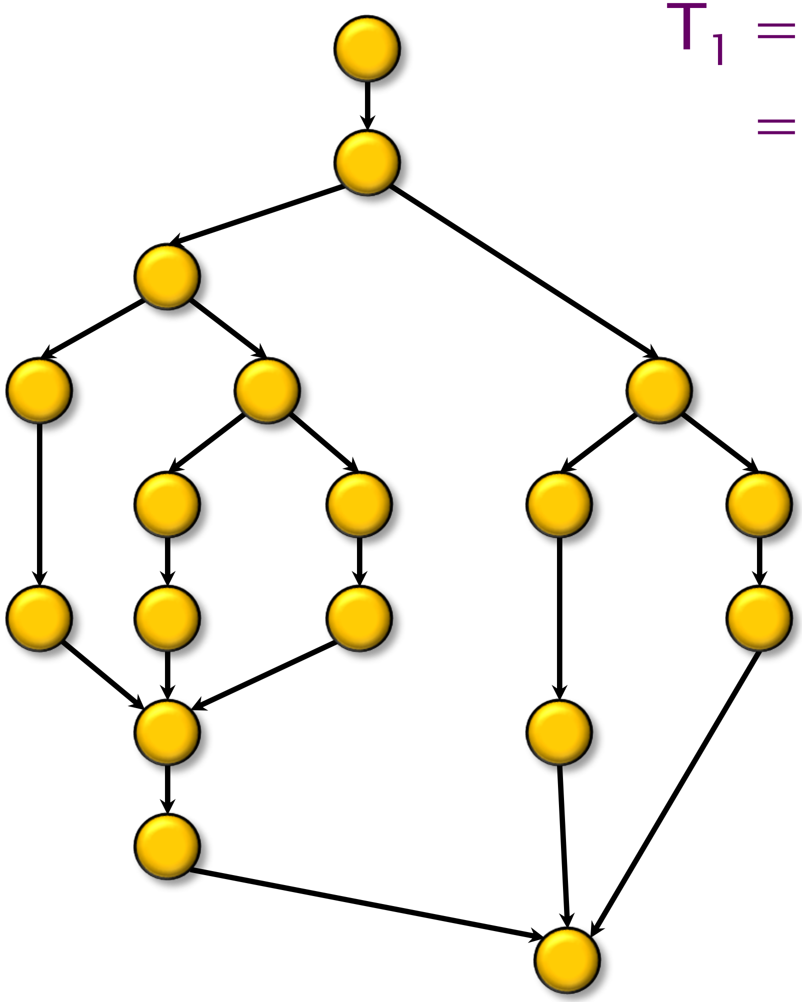
What is the parallelism of this computation?



Amdahl's Law says that since the serial fraction is $3/18 = 1/6$, the speedup is upper-bounded by 6.

Performance Measures

T_P = execution time on P processors

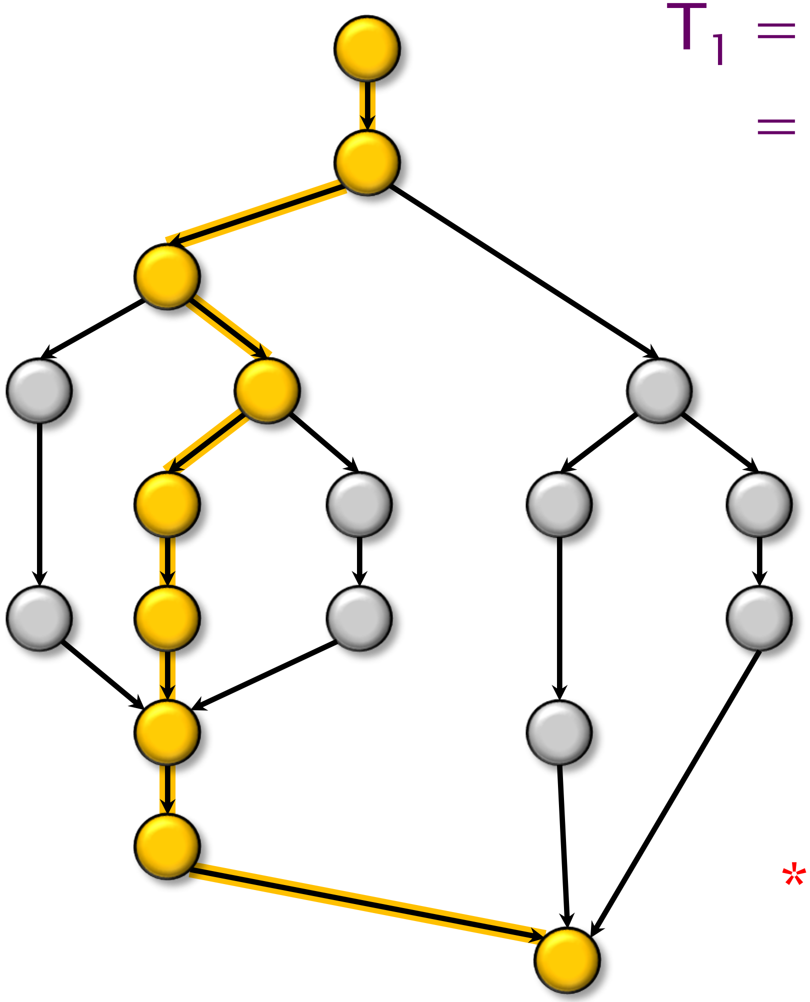


T_1 = work

$= 18$

Performance Measures

T_P = execution time on P processors



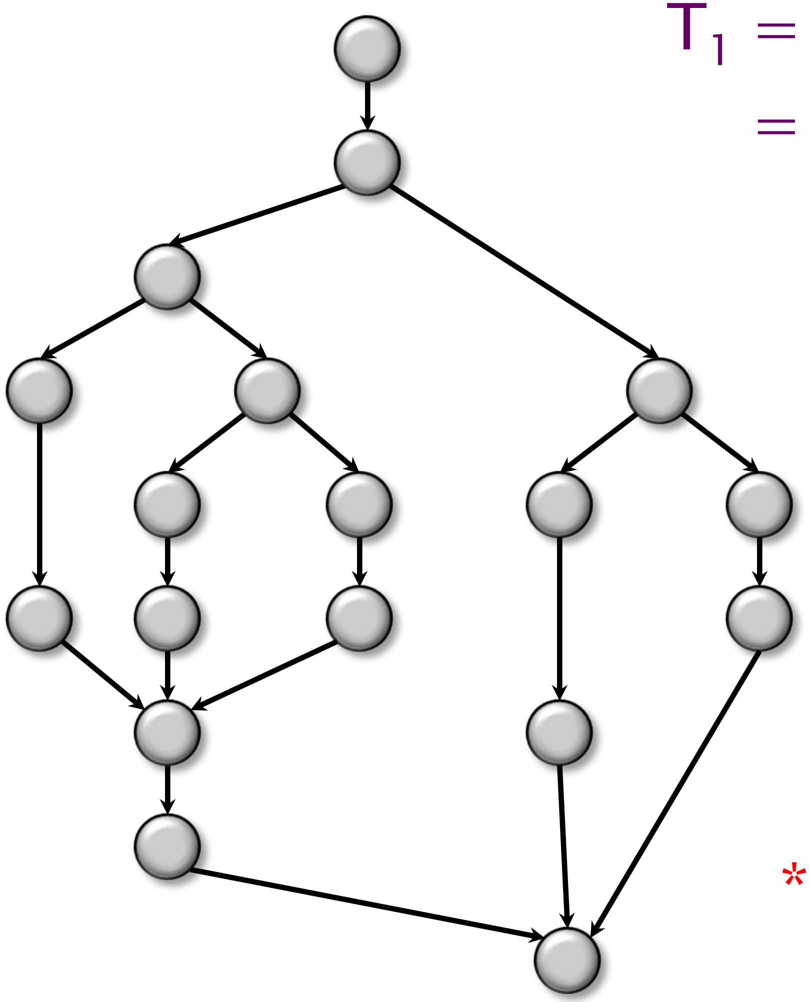
T_1 = work
= 18

T_∞ = span*
= 9

*Also called critical-path length or computational depth.

Performance Measures

T_P = execution time on P processors



T_1 = work
= 18

T_∞ = span*
= 9

WORK LAW
- $T_P \geq T_1/P$

SPAN LAW
- $T_P \geq T_\infty$

*Also called critical-path length or computational depth.

Speedup

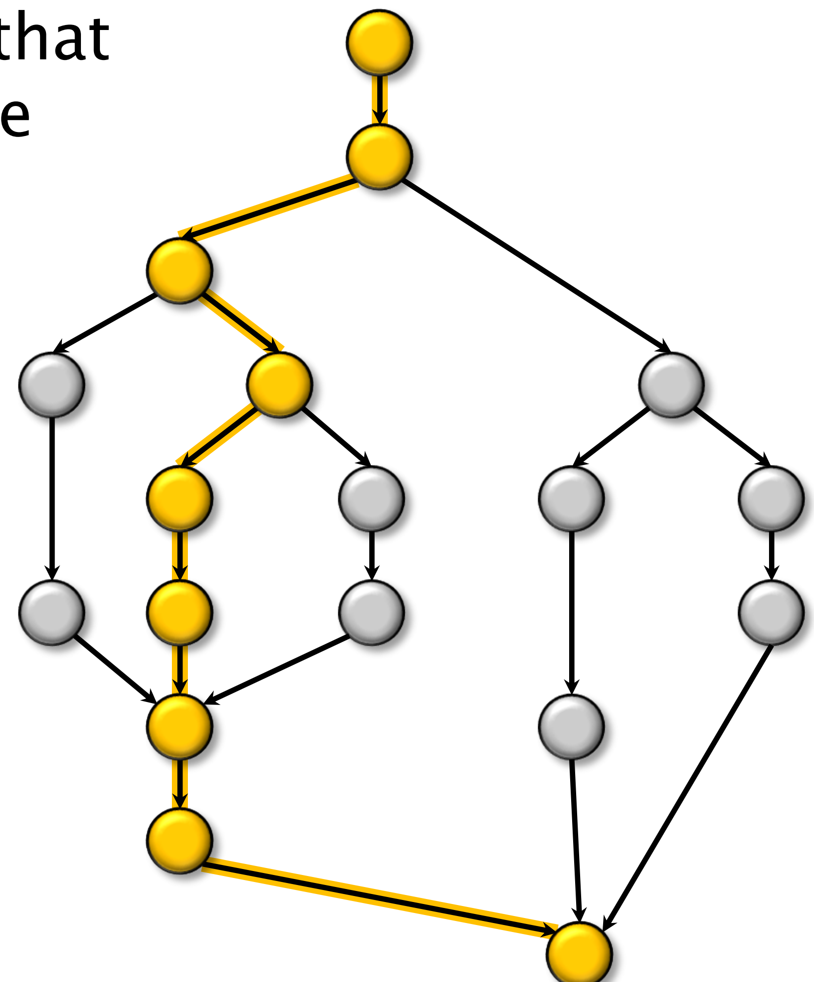
Definition. T_1/T_P = speedup on P processors.

- If $T_1/T_P < P$, we have sublinear speedup.
- If $T_1/T_P = P$, we have (perfect) linear speedup.
- If $T_1/T_P > P$, we have superlinear speedup, which is not possible in this simple performance model, because of the WORK LAW $T_P \geq T_1/P$.

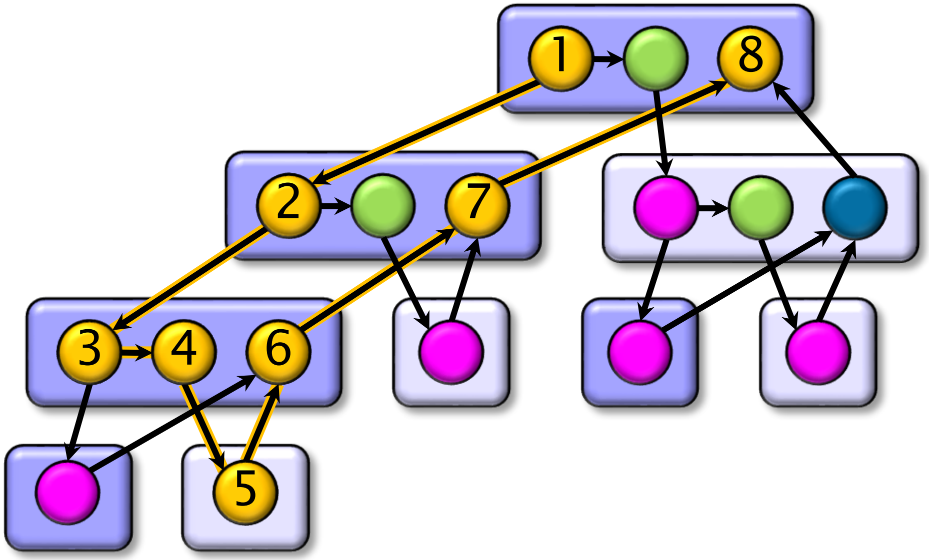
Parallelism

Because the SPAN LAW dictates that $T_P \geq T_\infty$, the maximum possible speedup given T_1 and T_∞ is

$$\begin{aligned} T_1/T_\infty &= \text{parallelism} \\ &= \text{the average amount of work per step along the span} \\ &= 18/9 \\ &= 2\,. \end{aligned}$$



Example: `fib(4)`



Assume for simplicity that each strand in `fib(4)` takes unit time to execute.

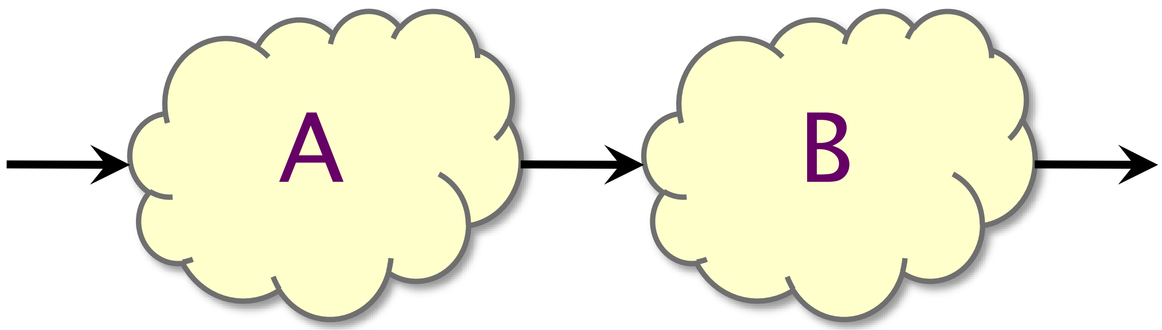
Work: $T_1 = 17$

Span: $T_\infty = 8$

Parallelism: $T_1/T_\infty = 2.125$

Using more than 2 processors guarantees that some processors will be idle.

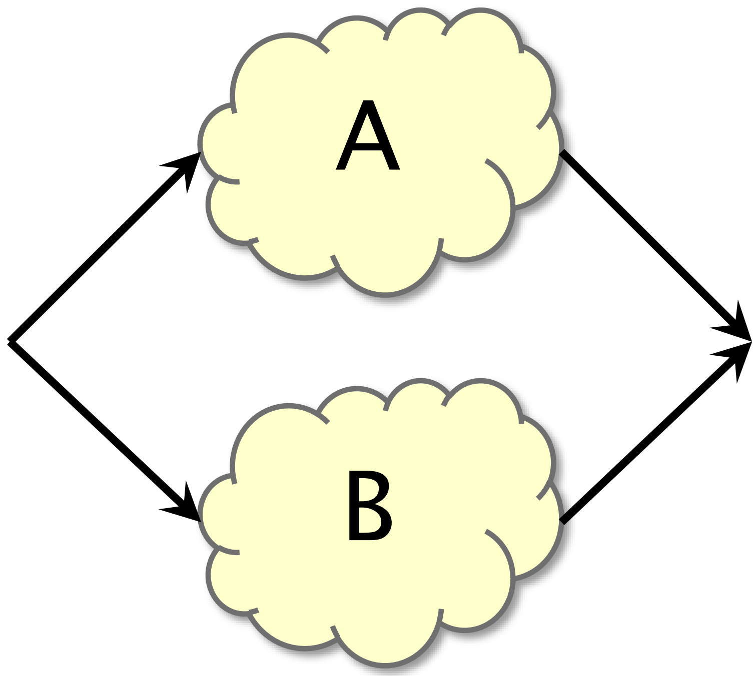
Quiz: Series Composition



Work: $T_1(A \cup B) = T_1(A) + T_1(B)$

Span: $T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$

Quiz: Parallel Composition



Work: $T_1(A \cup B) = T_1(A) + T_1(B)$

Span: $T_\infty(A \cup B) = \max\{T_\infty(A), T_\infty(B)\}$

HANDS-ON: THE CILKSCALE SCALABILITY ANALYZER

Quicksort Analysis

Example: Parallel quicksort

```
void quicksort(int64_t *left, int64_t *right)
{
  int64_t *middle;
  if (left == right) return;
  middle = partition(left, right);
  cilk_spawn quicksort(left, middle);
  quicksort(middle + 1, right);
  cilk_sync;
}
```

Analyze the sorting of 1,000,000 numbers.

★ ★ ★ *Guess the parallelism!* ★ ★ ★

Cilkscale Scalability Analyzer

- The Tapir/LLVM compiler provides a scalability analyzer called Cilkscale.
- Like the Cilksan race detector, Cilkscale uses compiler-instrumentation to analyze a serial execution of a program.
- Cilkscale computes work and span to derive upper bounds on parallel performance.

Run Cilkscale on QSort

1. Use make to compile `qsort`:

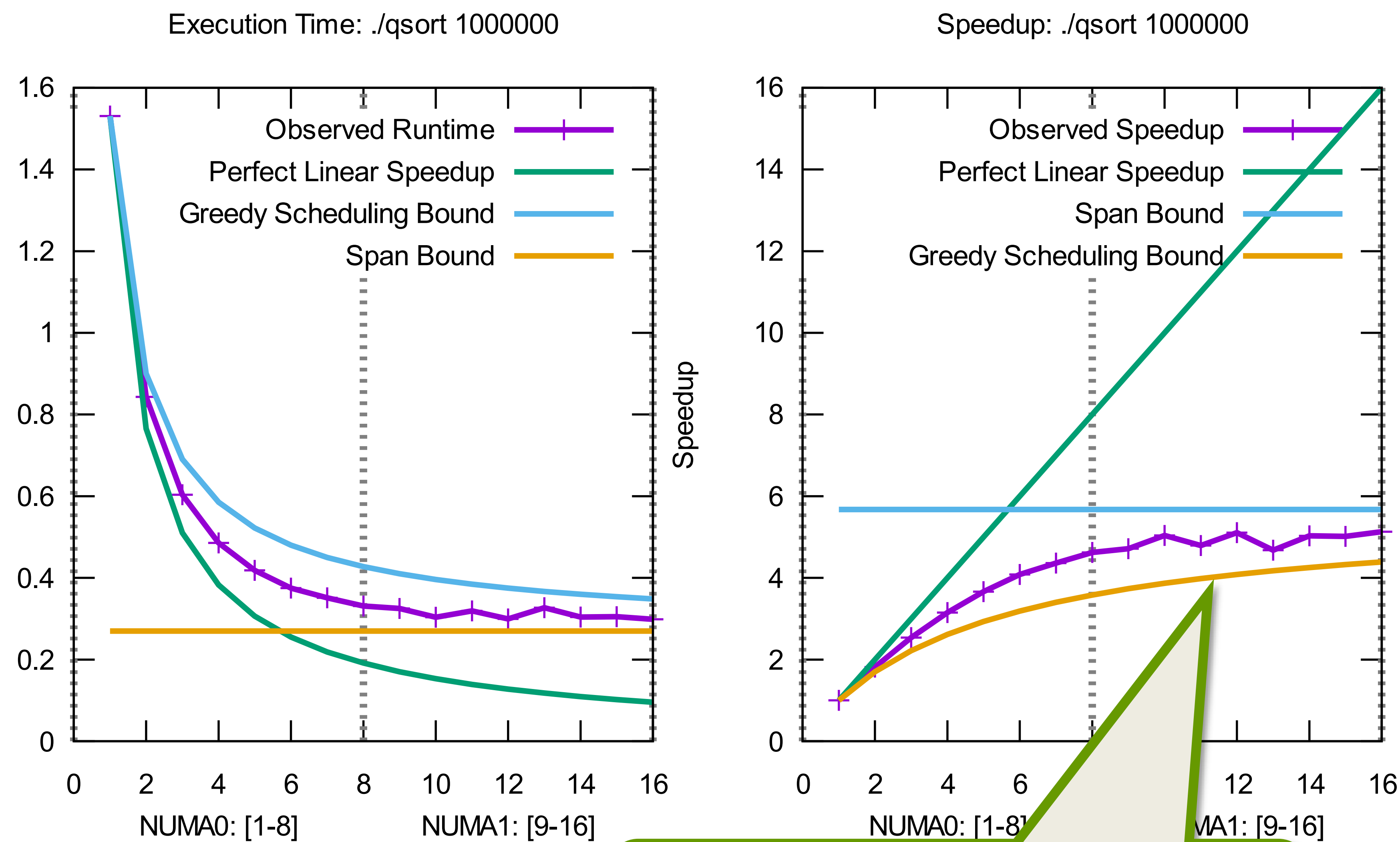
```
$ cd qsort
$ make
```

2. Run the Cilkscale script on `qsort` of 1,000,000 elements:

```
$ cilkscale ./qsort 1000000
```

3. Browse the files in your Jupyter notebook to find the output file `qsort-1000000.svg`, and open that file in your browser:
   - Click the checkbox next to the file.
   - Click the “View” button at the top of the window.

Cilkscale Output



We will derive this bound in the next section.

Theoretical Analysis

Example: Parallel quicksort

```
void quicksort(int64_t *left, int64_t *right)
{
  int64_t *middle;
  if (left == right) return;
  middle = partition(left, right);
  cilk_spawn quicksort(left, middle);
  quicksort(middle + 1, right);
  cilk_sync;
}
```

Expected work $= O(n \lg n)$
Expected span $= \Omega(n)$

⇒ Parallelism $= O(\lg n)$

Interesting Practical* Algorithms

Algorithm	Work	Span	Parallelism
Merge sort	$\Theta(n \lg n)$	$\Theta(\lg^3 n)$	$\Theta(n/\lg^2 n)$
Matrix multiplication	$\Theta(n^3)$	$\Theta(\lg n)$	$\Theta(n^3/\lg n)$
Strassen	$\Theta(n^{\lg 7})$	$\Theta(\lg^2 n)$	$\Theta(n^{\lg 7}/\lg^2 n)$
LU-decomposition	$\Theta(n^3)$	$\Theta(n \lg n)$	$\Theta(n^2/\lg n)$
Tableau construction	$\Theta(n^2)$	$\Theta(n^{\lg 3})$	$\Theta(n^{2-\lg 3})$
FFT	$\Theta(n \lg n)$	$\Theta(\lg^2 n)$	$\Theta(n/\lg n)$
Breadth-first search	$\Theta(E)$	$\Theta(D \lg V)$	$\Theta(E/D \lg V)$

*Cilk on 1 processor competitive with the best C.