# The dynamic multithreading model (part 1)

CSE 6230, Fall 2014

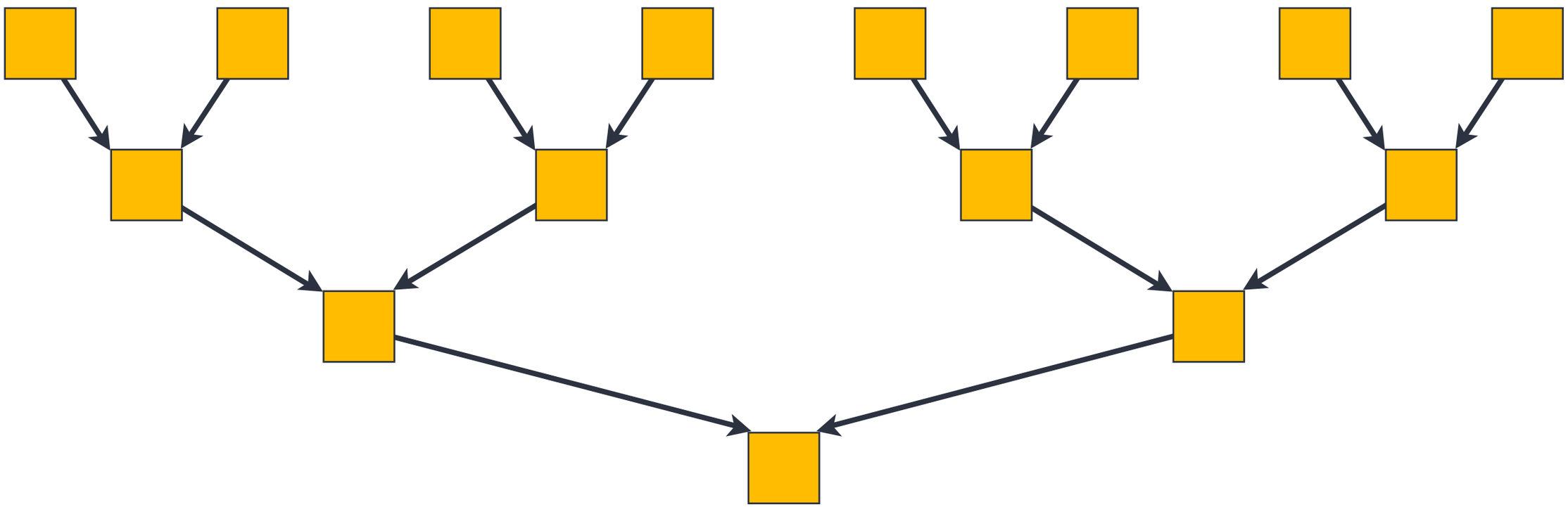
August 26

Georgia Tech College of Computing

Computational Science and Engineering

# Recall: DAG model of parallel computation

- **Work** = Total ops. Could interpret as sequential time.
- **Span** (or **depth**) = Length of longest seq. dependence chain.
- Example: **W** = 15, **D** = 4. **“Available” parallelism** = **W** / **D** = 3.75



# Relating work, depth, and time on *p* "ideal" processors

- Work & span laws

$$T_p \geq \frac{W}{p} \qquad T_p \geq D$$

- Speedup & ideal speedup

$$S_p \equiv \frac{T_1}{T_p} \leq p$$

- Brent's theorem

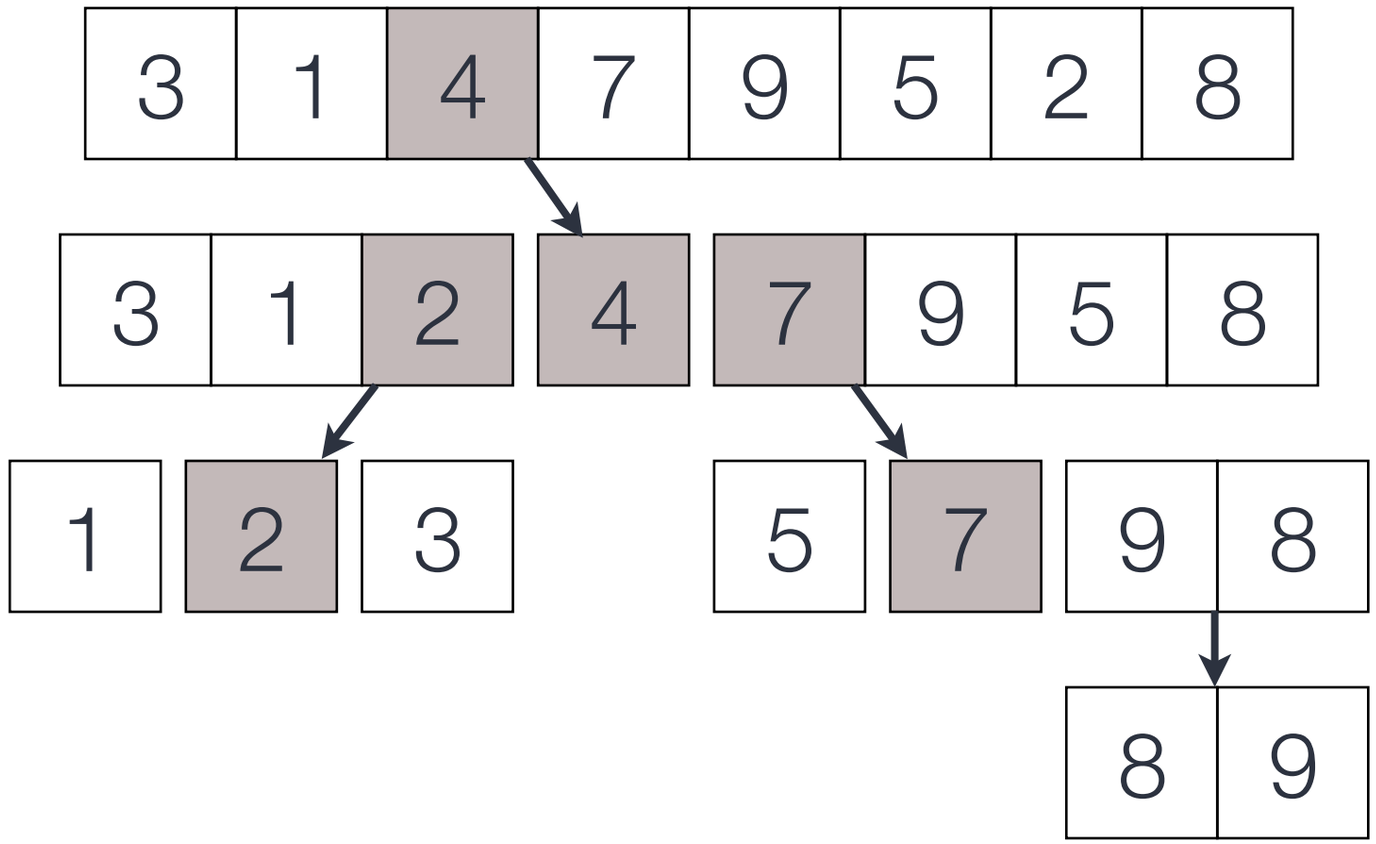
$$T_p \leq D + \frac{W - D}{p}$$

# Parallel primitives to generate DAGs

- A **dynamic multithreading** model
  - Augment the usual sequential model with three concurrency keywords: **spawn**, **sync**, **parallel-for**
  - Generates **nested data-parallel** DAGs
  - Permits “simple” analysis of work, depth
  - See new “readings” link at website for PDF file
- Data-parallel operations, e.g., **vector-add**, **scan**

# Quicksort

"Natural" parallelism exists at each branch in the recursion.



1: function $Y \leftarrow \text{qsort}(X)$ // $|X| = n$
2: if $|X| \leq 1$ then
3: return $Y \leftarrow X$
4: else
5: $X_L, Y_M, X_R \leftarrow \text{partition-seq}(X)$ **// Pivot**
6: $Y_L \leftarrow \text{qsort}(X_L)$
7: $Y_R \leftarrow \text{qsort}(X_R)$
8: return $Y \leftarrow Y_L \cup Y_M \cup Y_R$
9: endif

# Quicksort

```
1: function Y ← qsort(X) // |X| = n
2: if |X| ≤ 1 then
3:     return Y ← X
4: else
5:     X_L, Y_M, X_R ← partition-seq(X) // Pivot
6:     Y_L ← spawn qsort(X_L)
7:     Y_R ← spawn qsort(X_R)
8:     return Y ← Y_L ∪ Y_M ∪ Y_R
9: endif
```

# Quicksort

```
1:  function Y ← qsort(X) // |X| = n
2:  if |X| ≤ 1 then
3:      return  Y ← X
4:  else
5:      X_L, Y_M, X_R ← partition-seq(X) // Pivot
6:      Y_L ← spawn qsort(X_L)
7:      Y_R ← spawn qsort(X_R)
8:      sync
9:      return  Y ← Y_L ∪ Y_M ∪ Y_R
10: endif
```

# Quicksort

```
1:  function Y ← qsort(X) // |X| = n
2:  if |X| ≤ 1 then
3:      return  Y ← X
4:  else
5:      X_L, Y_M, X_R ← partition-seq(X) // Pivot
6:      Y_L ← spawn qsort(X_L)
7:      Y_R ← qsort(X_R)
8:      sync
9:      return  Y ← Y_L ∪ Y_M ∪ Y_R
10: endif
```

# Quicksort

```
1: for i ← 1 to n do
2:     f(i)
```

# Parallel loops

```
1: parallel-for i ← 1 to n do
2:     f(i)
```

# Parallel loops

```
1: parallel-for i ← 1 to n do
2:     f(i)
```

Work and span?

# Parallel loops