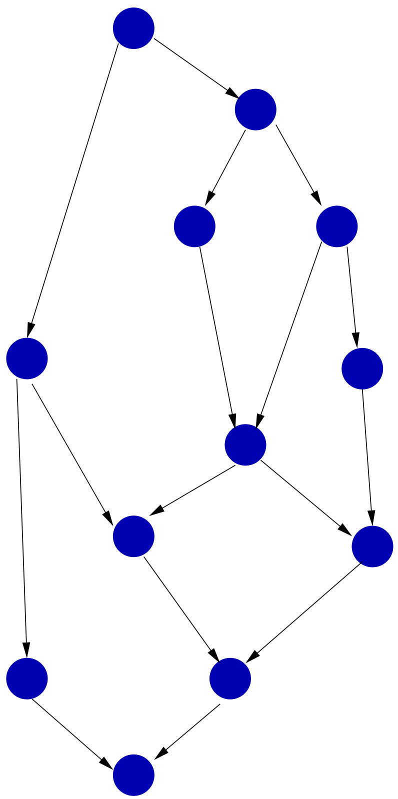
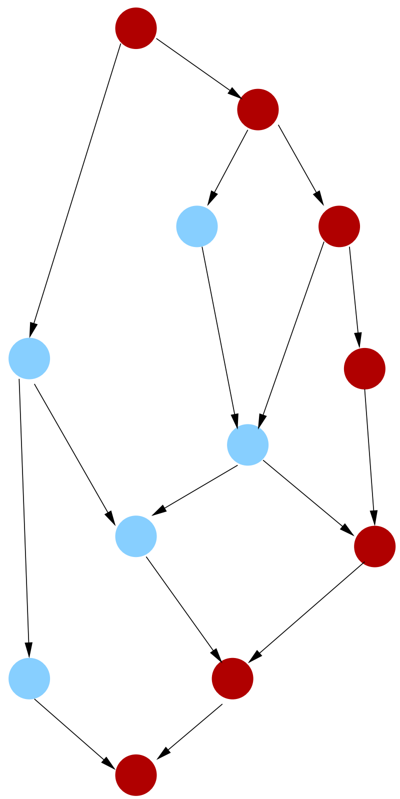
# Work and Span

work
12





span
7

## Amdahls Law with Work and Span

$T = work$ = sequential time    $T_p$ = wall-clock time $p$ CPUs    $T_\infty$ = wall-clock time $\infty$ CPUs

Speedup  $S_P = T/T_P$

$span$ critical path  (also called “makespan” in the context of scheduling)

$f$ fraction of sequential work, thus  $f = span/work$

simplified Amdahl’s law in terms of $work$ and $span$:  $S_p \leq 1/f = work/span$

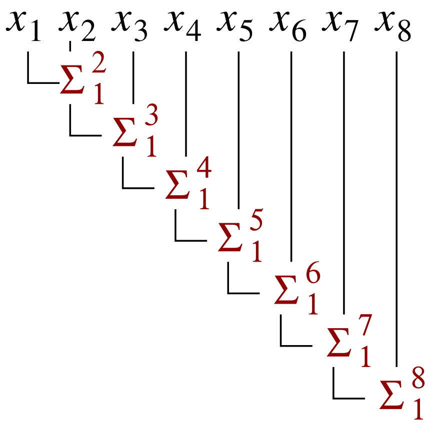
Reduce $span$ as much as possible:

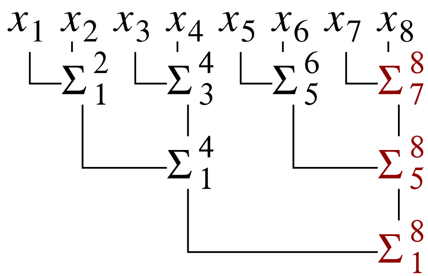
- ☐ keep sequential blocks short! $\Rightarrow$ coarse grained locking is evil
- ☐ keep sequential dependencies short! $\Rightarrow$ (non-logarithmic) loops are evil

## Sum

compute sum $\sum_1^n x_i$ for $n$ numbers $x_i$ in parallel

- sequential
  - $y_0 = 0, \quad y_{i+1} = y_i + x_i$ for $i = 1 \dots n-1$
  - $work = T = \mathcal{O}(n) \quad (n-1$ additions)
  - $span = \mathcal{O}(n)$ too
  - since $y_{i+1}$ depends on all previous $y_j$ with $j \leq i$
  - thus no speed-up $S_p = \mathcal{O}(1)$
- parallel
  - **associativity** allows to regroup computation
  - $work = \mathcal{O}(n)$ remains the same
  - $span = \mathcal{O}(\log n)$ reduces exponentially
  - speed-up not ideal but $S_n = \mathcal{O}(n/\log n)$
  - note $p > n$ does not make sense



## Prefix / Scan

compute all sums $s_j = \sum_1^j x_i$ for all $j = 1 \ldots n$ and again $n$ numbers $x_i$ in parallel

sequential version as in previous slide

parallel version needs a second depth $\mathcal{O}(\log\, n)$ pass

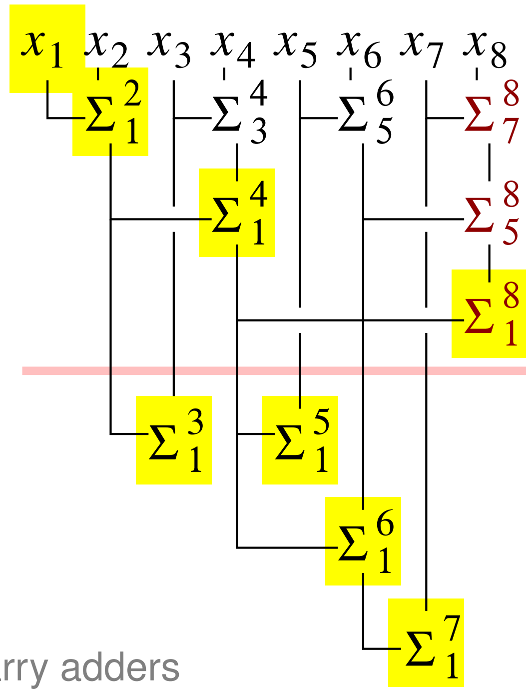
works even "in place" (first pass overwrites original $x_i$)

but actual "wiring" complicated

still $span = \mathcal{O}(\log\, n)$

basic algorithmic idea for many "parallel" algorithms

propagate and generate adders with prefix trees instead of ripple carry adders

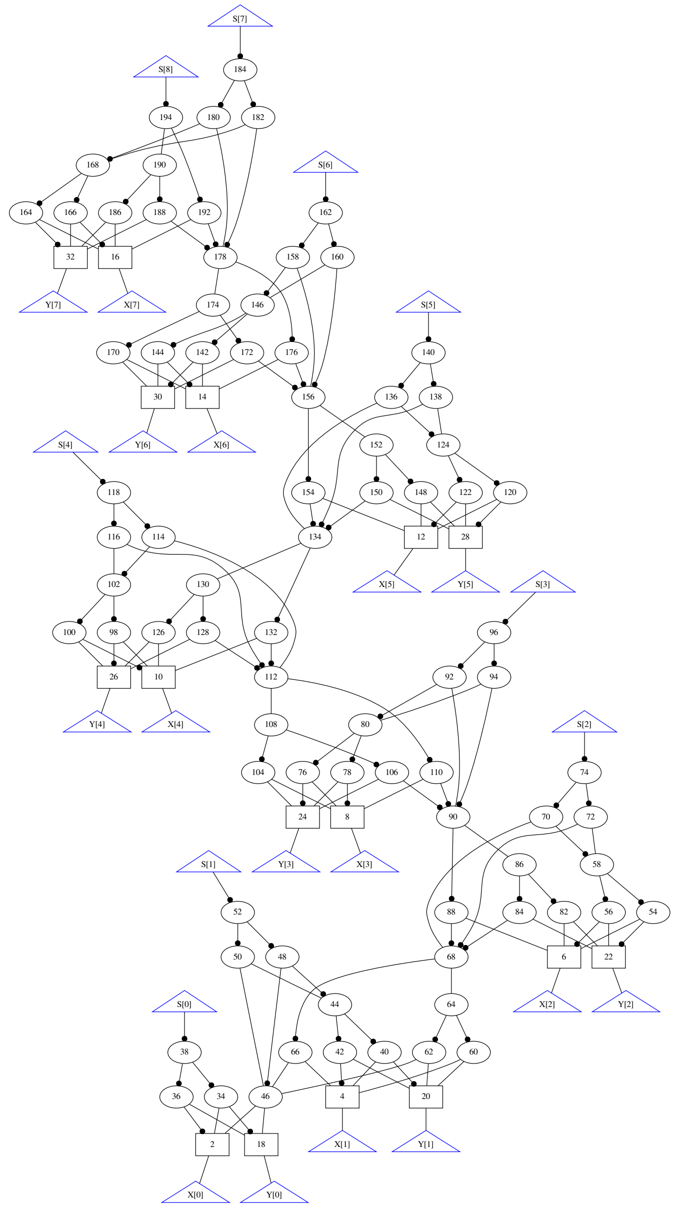


# Ripple-Carry-Adder

$$
\begin{aligned}
s_i &= x_i \oplus y_i \oplus c_i & & \text{sum} \\
c_{i+1} &= x_i\, y_i + x_i\, c_i + y_i\, c_i & & \text{carry}
\end{aligned}
$$

$$
\begin{aligned}
& & c_0 &= 0 \\
s_0 &= x_0 \oplus y_0 & c_1 &= x_0\, y_0 \\
s_1 &= x_1 \oplus y_1 \oplus c_1 & c_2 &= x_1\, y_1 + x_1\, c_1 + y_1\, c_1 \\
s_2 &= x_2 \oplus y_2 \oplus c_2 & c_3 &= x_2\, y_2 + x_2\, c_2 + y_2\, c_2 \\
s_3 &= x_3 \oplus y_3 \oplus c_3 & c_4 &= x_3\, y_3 + x_3\, c_3 + y_3\, c_3 \\
s_4 &= x_4 \oplus y_4 \oplus c_4 & c_5 &= x_4\, y_4 + x_4\, c_4 + y_4\, c_4 \\
s_5 &= x_5 \oplus y_5 \oplus c_5 & c_6 &= x_5\, y_5 + x_5\, c_5 + y_5\, c_5 \\
s_6 &= x_6 \oplus y_6 \oplus c_6 & c_7 &= x_6\, y_6 + x_6\, c_6 + y_6\, c_6 \\
s_7 &= x_7 \oplus y_7 \oplus c_7 & c_8 &= x_7\, y_7 + x_7\, c_7 + y_7\, c_7
\end{aligned}
$$

$$
work = \mathcal{O}(n) \qquad span = \mathcal{O}(n)
$$



# Propagate-and-Generate Adder / Lookahead Adder

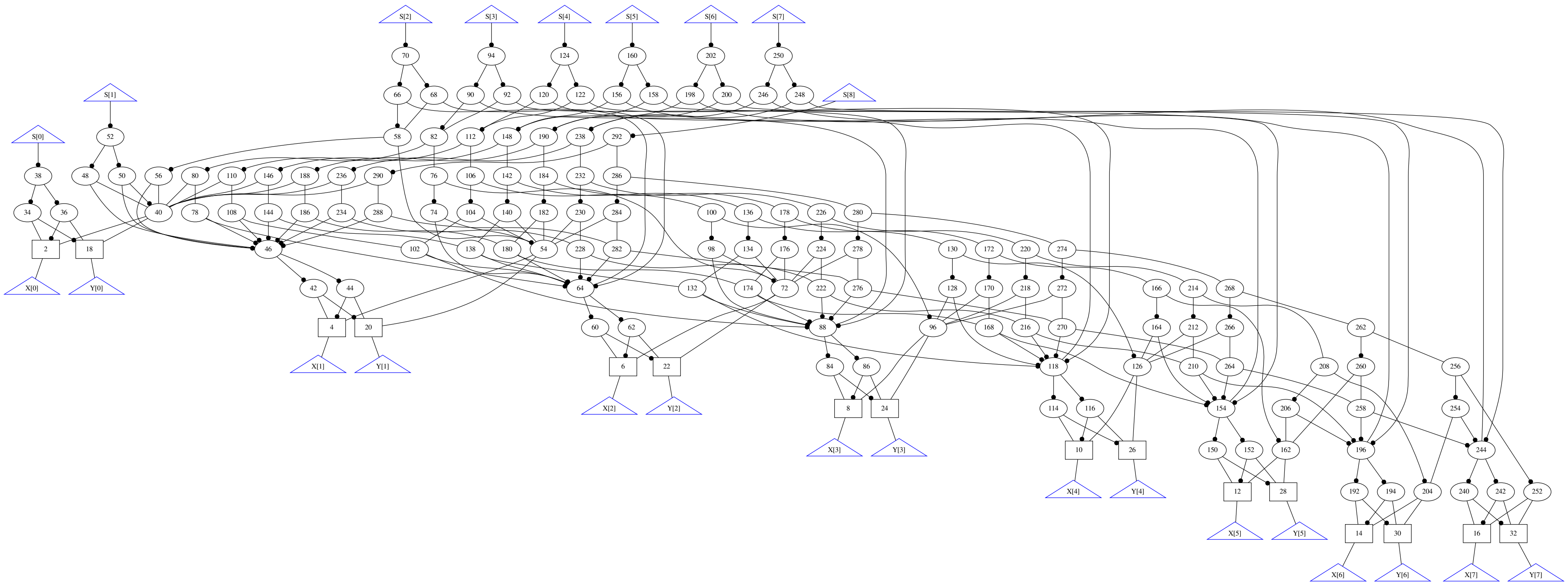
$$
\begin{aligned}
p_i &= x_i + y_i && \text{propagate} \\
g_i &= x_i \, y_i && \text{generate}
\end{aligned}
$$

$$
c_{i+1} = g_i + p_i \, c_i \qquad \text{new carry computation formula}
$$

$$
\begin{aligned}
c_0 &= 0 \\
c_1 &= g_0 \\
c_2 &= g_1 + p_1 \, g_0 \\
c_3 &= g_2 + p_2 \, g_1 + p_2 \, p_1 \, g_0 \\
c_4 &= g_3 + p_3 \, g_2 + p_3 \, p_2 \, g_1 + p_3 \, p_2 \, p_1 \, g_0 \\
c_5 &= g_4 + p_4 \, g_3 + p_4 \, p_3 \, g_2 + p_4 \, p_3 \, p_2 \, g_1 + p_4 \, p_3 \, p_2 \, p_1 \, g_0 \\
c_6 &= g_5 + p_5 \, g_4 + p_5 \, p_4 \, g_3 + p_5 \, p_4 \, p_3 \, g_2 + p_5 \, p_4 \, p_3 \, p_2 \, g_1 + p_5 \, p_4 \, p_3 \, p_2 \, p_1 \, g_0 \\
c_7 &= g_6 + \ldots + \ldots \, p_6 \, p_5 \, p_4 \, p_3 \, p_2 \, p_1 \, g_0 \\
c_8 &= g_7 + \ldots + \ldots \, p_7 \, p_6 \, p_5 \, p_4 \, p_3 \, p_2 \, p_1 \, g_0
\end{aligned}
$$

$work = \mathcal{O}(n^2)$ $\quad span = \mathcal{O}(\log\, n)$ $\qquad$ assuming $n$-ary gates otherwise $\quad work = \mathcal{O}(n^3)$

# Carry-Lookahead Adder




$work = \mathcal{O}(n^2) \quad span = \mathcal{O}(\log n)$

using prefix / scan computation otherwise work remains $\mathcal{O}(n^3)$ for binary AND gates