Exercise 3

Introduction to High-Performance Computing

WS 2019/20

Simon Schwitanski

Christian Terboven

contact@hpc.rwth-aachen.de

Chair for High Performance Computing, RWTH Aachen University

Exercise Tasks

1. **Scaling behavior**
2. **Cache coherence**
3. **Key figures of networks**
4. **Task dependency graphs**

1. Scaling behavior
1.1. Strong scaling

- **Given: parallel execution time with 4 processes = 12h**
  10% is serial processing

→ $T(4) = 12h,\ s_{4procs} = 0.1$ (s for 4 processors!)

→ Serial processing time is $0.1 \cdot 12h = 1.2h$

→ Parallel processing time is $10.8h$

a) Calculate compute time on 16 parallel processes!

→ $T(16) = s_{4procs} \cdot T(4) + p_{4procs} \cdot \frac{T(4)}{4} = 1.2h + \frac{10.8h}{4} = 3.9h$

(Caution: Baseline was given with 4 procs., so we divide by 4 instead of 16!)

b) How long takes serial execution? What is the sequential portion?

→ Serial execution: $T(1) = s_{4procs} \cdot T(4) + 4 \cdot p_{4procs} \cdot T(4) = 44.4h$

→ Sequential portion: $s_{1proc} = \frac{s_{4procs} \cdot T(4)}{T(1)} = \frac{1.2h}{44.4h} = 2.7\ \% = 0.027$

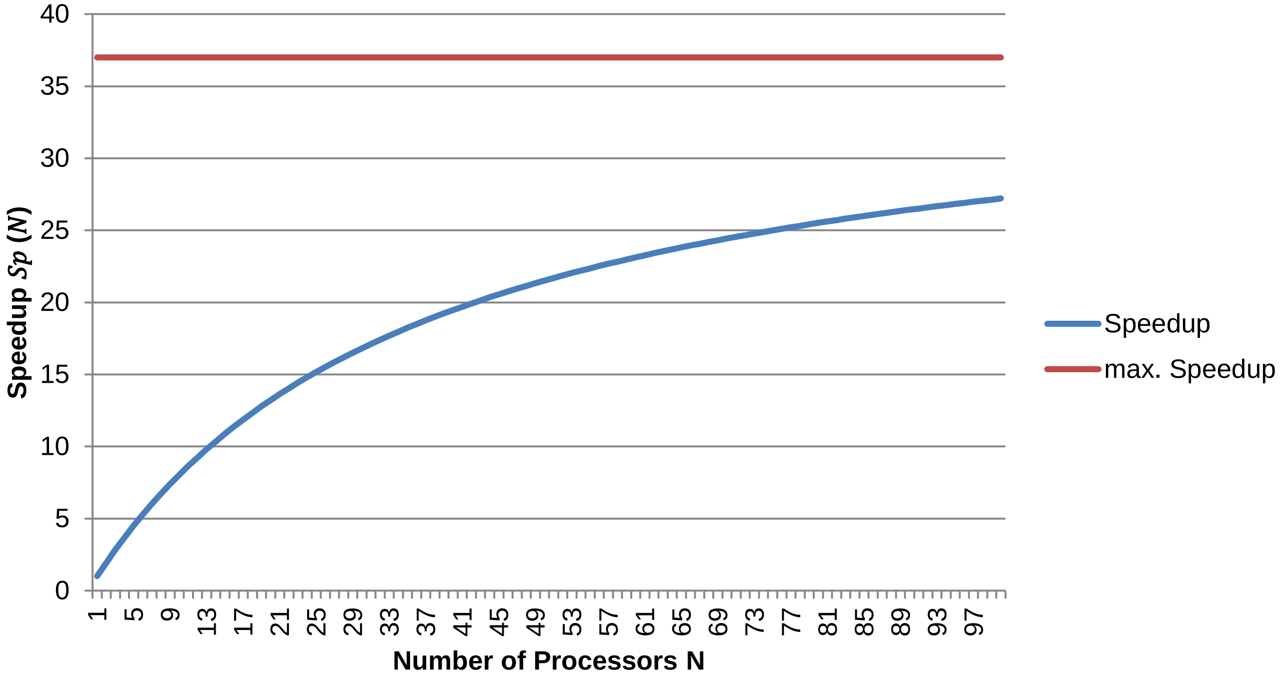
c) How many processes are needed to get a 2-hour execution time? What is the lower bound of parallel calculation time?

- → Intuitively:
  - → 2h total runtime - 1.2h sequential runtime = 0.8h parallel runtime
  - → 43.2h parallel workload / (0.8h runtime per process) = 54 processes
- → Formally:
  - → $T(N) = 2h$
  - → $T(N) = s \cdot T(1) + \frac{p \cdot T(1)}{N} \Rightarrow N = \frac{p \cdot T(1)}{T(N) - s \cdot T(1)} = \frac{43.2h}{0.8h} = 54$
- → Lower bound of parallel calculation time $T(N)$
  - → For $N \to \infty$: runtime $\to$ sequential runtime $= 1.2\text{h}$
  - → $\lim_{N \to \infty} T(N) = s \cdot T(1) + \frac{p \cdot T(1)}{N} = s \cdot T(1) = 1.2h$

d) Speedup based on serial runtime

$$\lim_{N\to\infty} S_p(N) = \lim_{N\to\infty} \frac{1}{s + \frac{1-s}{N}} = \frac{1}{s}$$

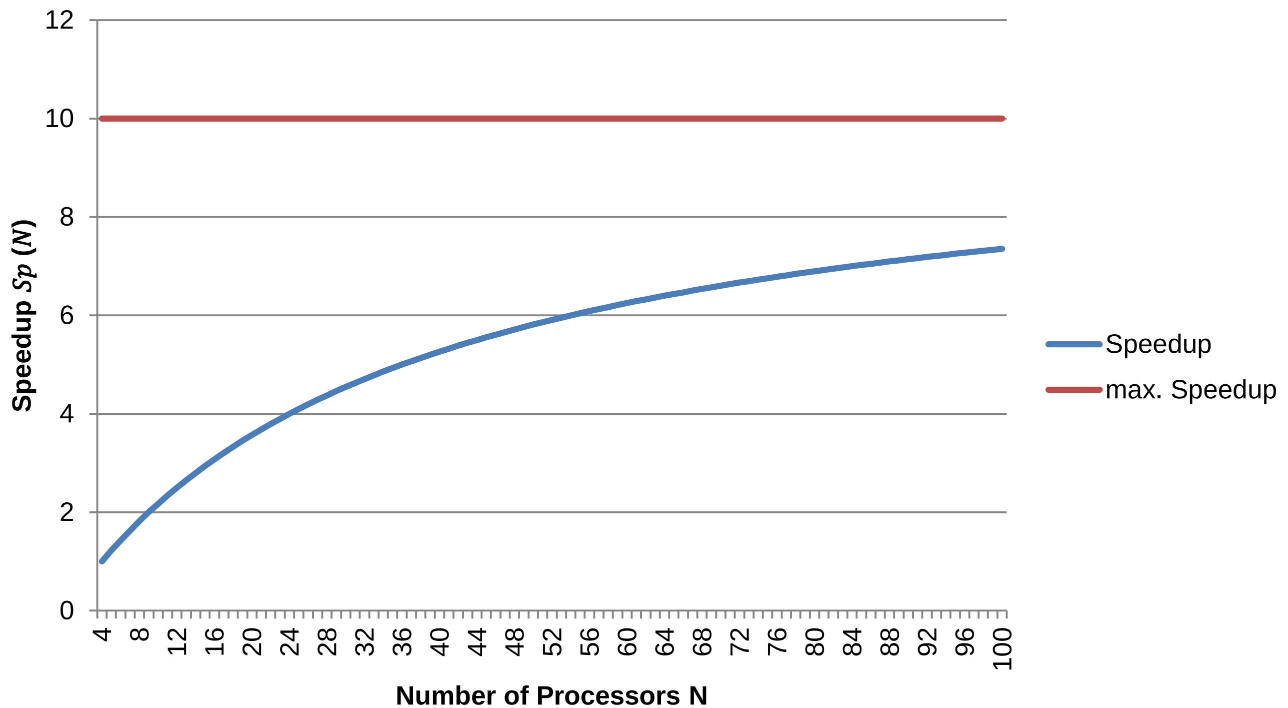
→ 1/0.027 =37



d) Speedup based on parallel execution with 4 processes

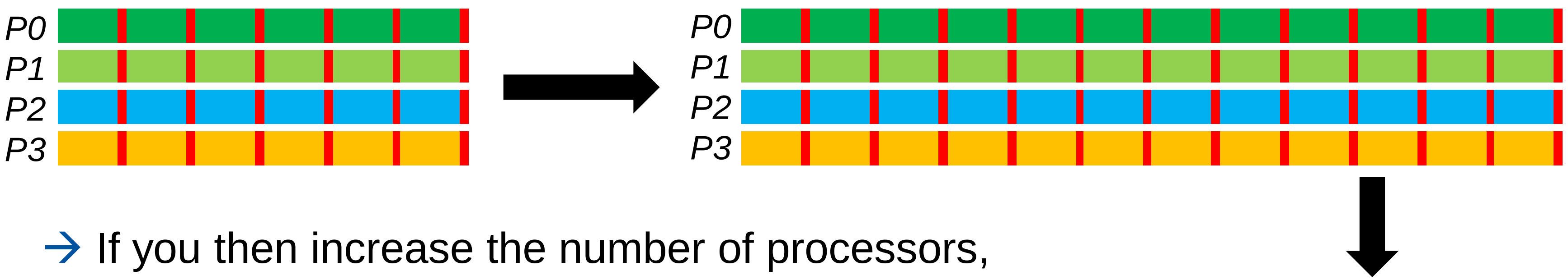
$$\lim_{N\to\infty} S_p(N) = \lim_{N\to\infty} \frac{1}{s + \frac{1-s}{N}} = \frac{1}{s}$$

→ 1/0.1 =10

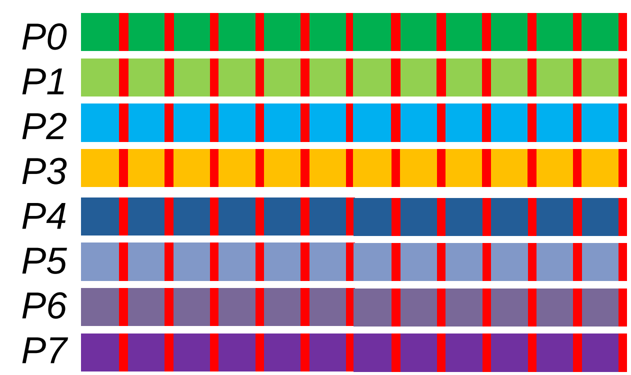


- **What parameter of a weather simulation could you adjust to have constant parallel computation time when you increase parallel computation capability?**
  - → Time resolution
  - → Spatial resolution
  - → Size of the region

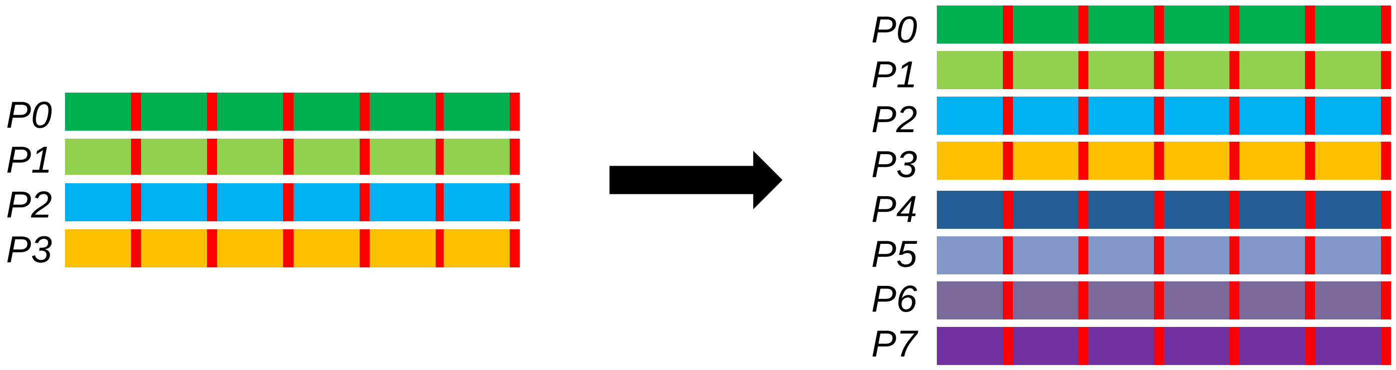
- **The basic strategy with weak scaling is to replicate workload for each process → fixed problem size per process**
- **Higher time resolution results in more iteration cycles:**
  - → If you increase the number of iteration cycles, you also add more calls to the sequential *update_grid* (red part) function (example: doubling iteration cycles)



  - → If you then increase the number of processors, then you decrease the time of each iteration cycle
  - → **But:** The sequential portion of the calculation gets bigger which contradicts Gustafson's assumption of having a constant sequential portion



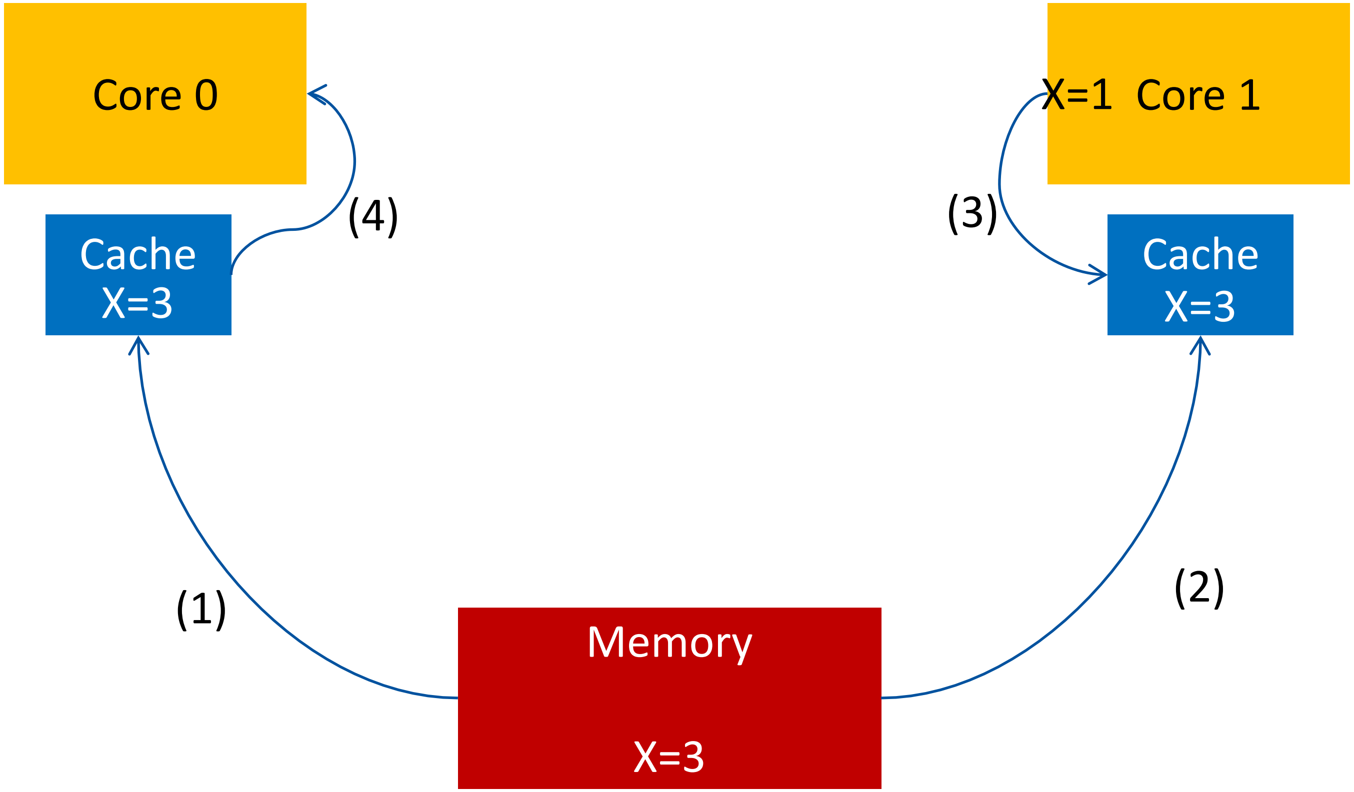
- **Higher spatial resolution and increasing the size of the region both result in more grid points.**
  - → Parallel workload increases with number of processes (e.g. doubling number of processes requires doubling number of grid points for same problem size per proc.)
  - → Time per iteration cycle stays constant with increasing number of processes and no further calls to *update_grid* are added (number of timesteps is fixed)



Exercise Tasks

1. Scaling behavior
2. **Cache coherence**
3. Key figures of networks
4. Task dependency graphs

Cache coherence



- After time step 3, cores see different values for X
- Depending on which cache writes back X, value might be stale

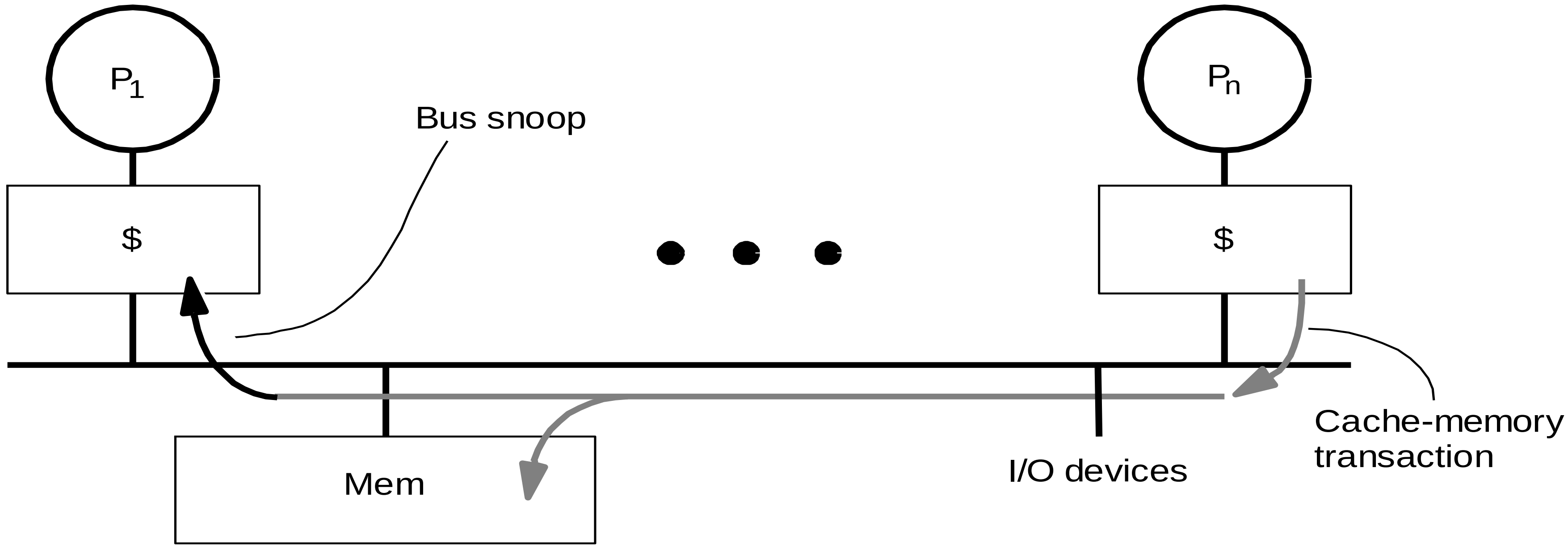
Cache coherence

- **When switching to multicore processors, the concept of caching needs some considerations:**
  - → Caches store
    - → Private data only used by a single core
    - → Data that is shared by multiple cores
  - → Sharing data inside a common cache
    - → Reduces memory latency on more than one core
    - → Increases the effective bandwidth of shared data
  - → Cache coherence problem
    - → Modifications of cached data must be visible to all cores

Strategies towards maintaining coherent caches

- **Directory based**
  - → Sharing status of a block of physical memory is kept in only one place, the directory
- **Snooping**
  - → Caches are connected via a broadcast medium and snoop(monitor) on that medium for memory blocks that they currently store

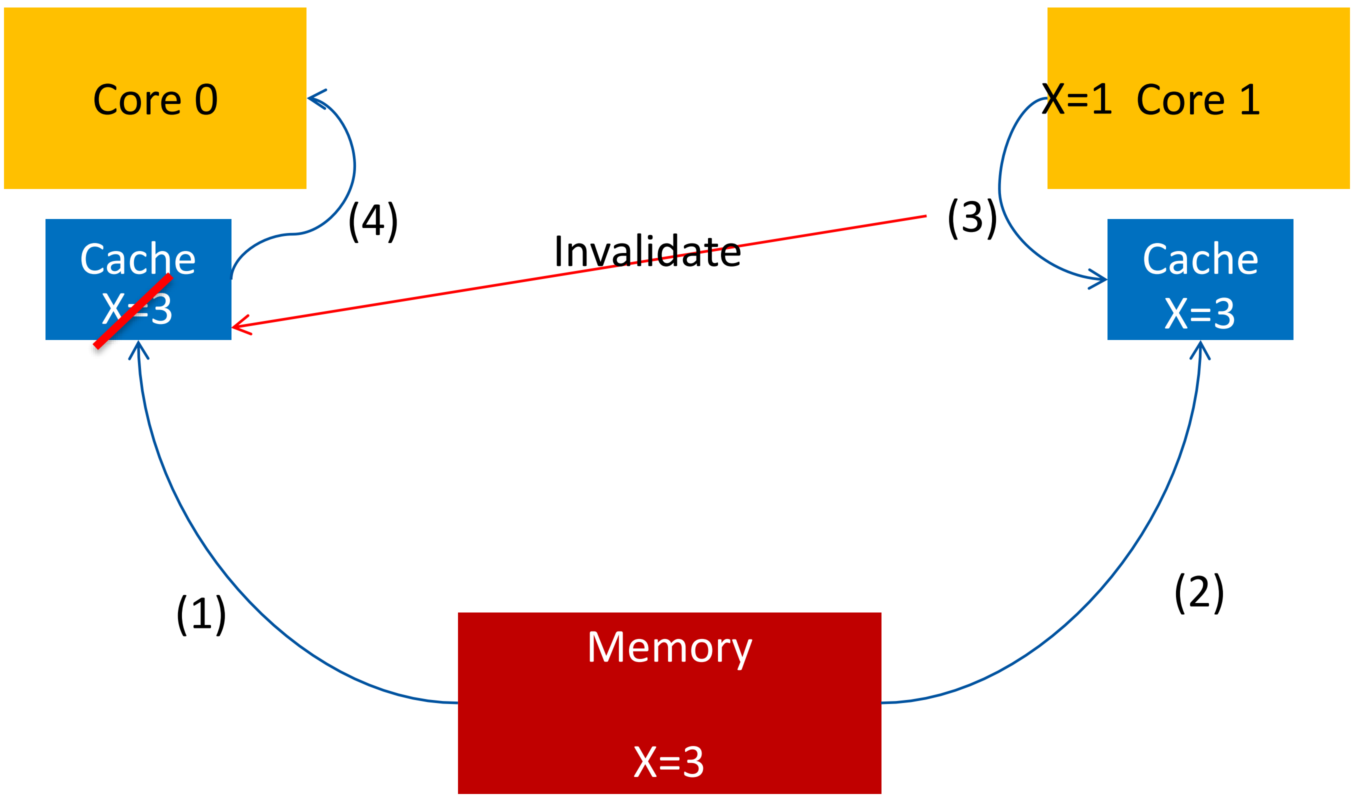
Cache coherence



- **Cache controller snoops on the medium and take relevant action to ensure cache coherence:**
  - → Invalidate
  - → Update
  - → Supply value

Source: TU Dresden, Daniel Molka

Cache coherence



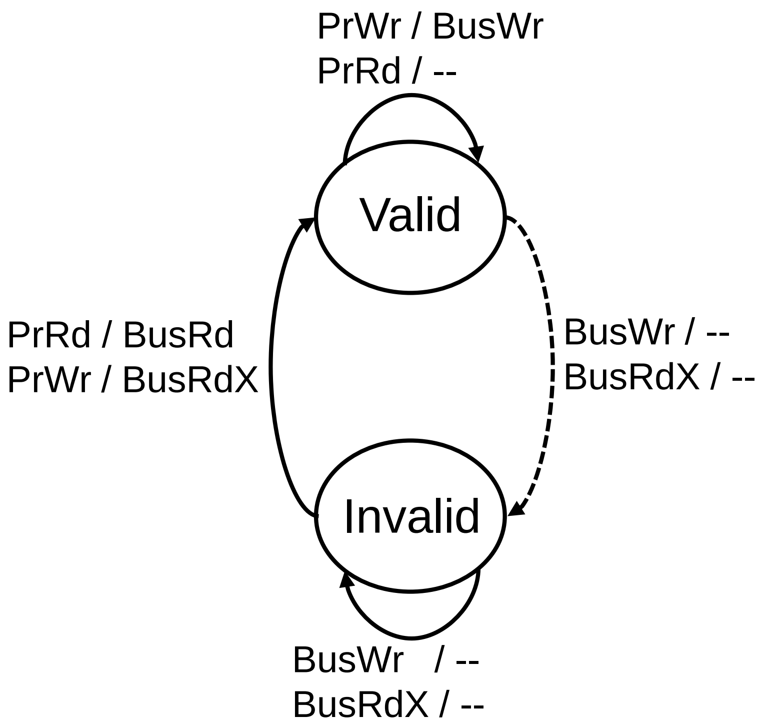
- **All recent MP systems use write-invalidate**

Cache coherence

- **Write-through**
  - → Data written to cache is directly written to memory
  - → Get most recent value from main memory
- **Write-back**
  - → Cache lines are not directly written to main memory but are flagged as dirty and written back later on
  - → All cores check their caches for addresses placed on the memory bus
  - → If core has most recent copy of requested cache block it provides it in response to a read request
- **Write-back vs. Write-through**
  - → Write-back needs lower memory bandwidth
  - → Most recent multiprocessors use write-back

Cache coherence protocols
Simple write-through protocol

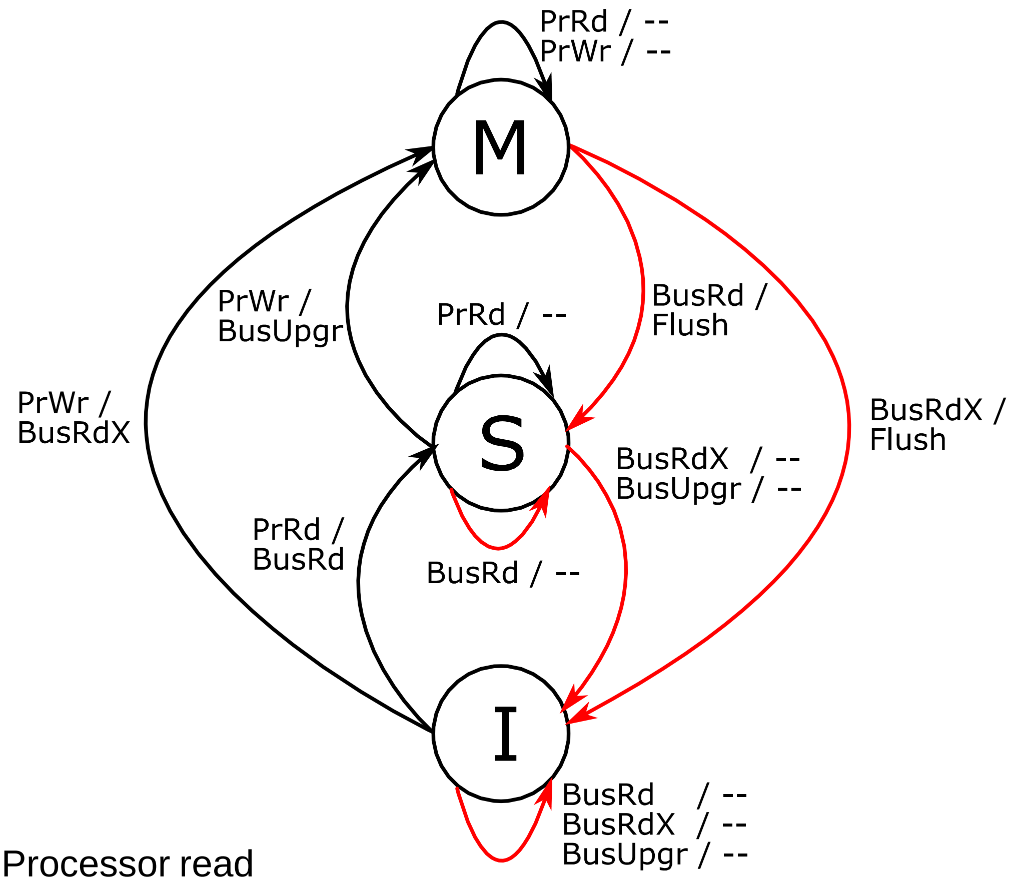
- **Write-through snooping protocol based on block invalidation**
- **Each block of memory has one of the following states**
  - → Valid: block is present in cache, all copies of block are identical to copy in main memory
  - → Invalid: block contains invalid data, needs the most updated copy from owner
    - → PrWr on invalid data requires a first a read



PrRd: Processor read
PrWr: Processor write
BusRd (bus read): read request for a block
BusRdX (bus read exclusive): read block and invalidate other copies
BusWr: write a block to memory and invalidate other copies

Cache coherence protocols
Example: MSI invalidate protocol

- **Write-back snooping protocol based on block invalidation**
- **Each block of memory has one of the following states**
  - → Modified: exactly one cache (owner) holds a valid copy, memory is stale
  - → Shared: zero or more other caches hold valid copy
  - → Invalid: block contains invalid data, needs the most updated copy from owner



PrRd: Processor read

PrWr: Processor write

BusRd (bus read): read request for a block

BusRdX (bus read exclusive): read block and invalidate other copies

BusUpgr: invalidate other copies

Flush: Write cache block back to main memory

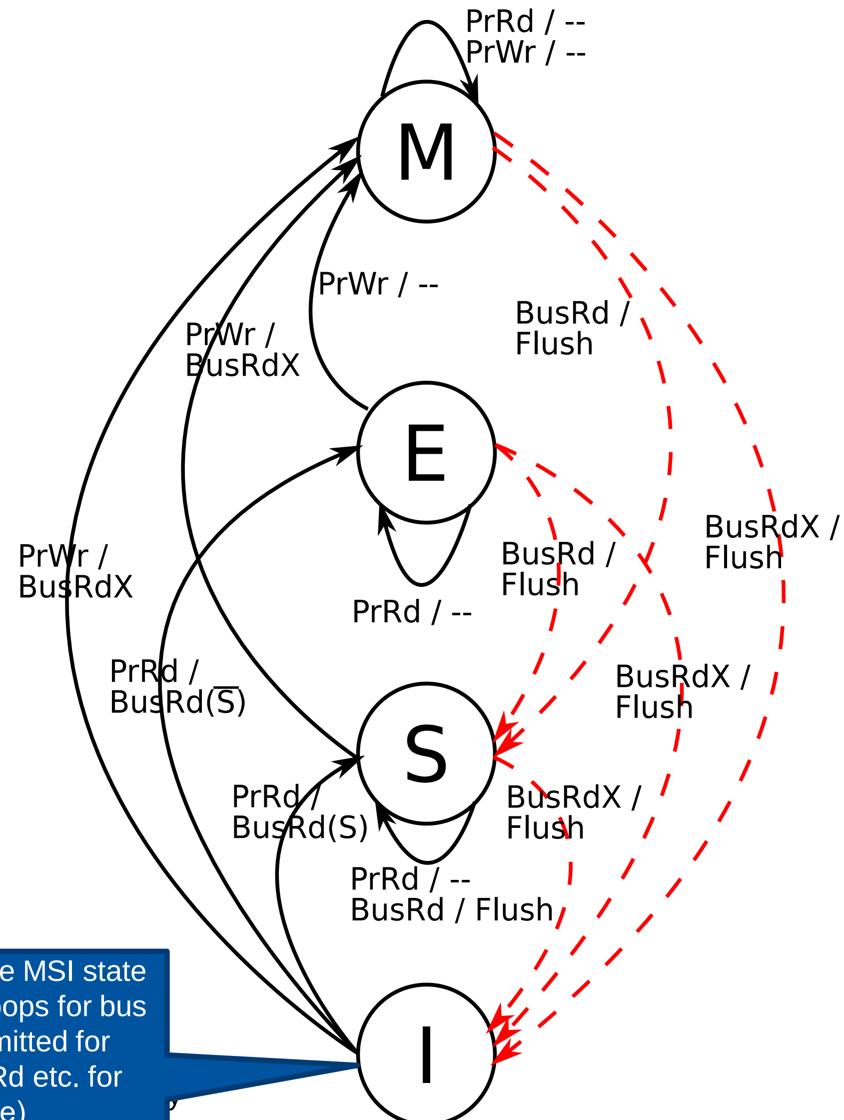
2. Cache Coherence
2.1 Applying the MSI Protocol

t	Local Request	P1	P2	P3	Gen. Bus Request	Data Supplier
0	Initially	-	-	-	-	-
1	R1	S	-	-	BusRd	Memory
2	W1	M	-	-	BusUpgr	-
3	R3	S	-	S	BusRd	P1's cache (flush)
4	W3	I	-	M	BusUpgr	-
5	R3	I	-	M	-	-
6	R2	I	S	S	BusRd	P3's cache (flush)

t	Local Request	P1	P2	P3	Gen. Bus Request	Data Supplier
0	Initially	-	-	-	-	-
1	W2	-	M	-	BusRdX	Memory
2	R2	-	M	-	-	-
3	R1	S	S	-	BusRd	P2's cache (flush)
4	W2	I	M	-	BusUpgr	-
5	R1	S	S	-	BusRd	P2's cache (flush)
6	R3	S	S	S	BusRd	P1/P2's cache (flush)
7	W1	M	I	I	BusUpgr	-
8	R1	M	I	I	-	-
9	W2	I	M	I	BusRdX	P1's cache
10	R3	I	S	S	BusRd	P2's cache

- **Add additional state "Exclusive"**
  - → Cache line has recently been read from memory but not yet modified. It does not reside in any other cache.
- **Additionally: Shared signal "S", determines on BusRd request if any other processor holds the cache block**
  - → BusRd(S): As BusRd, in addition: There is a copy of the corresponding cache block on at least one other processor.
  - → BusRd($\overline{S}$): As BusRd, in addition: There is **<u>no</u>** copy of the corresponding cache block on any other processor.
  - → If we write just "BusRd", we do not care about the shared signal "S".

MESI Protocol State Machine



Note: Compared to the MSI state machine, some self loops for bus transactions are omitted for simplicity (e.g. BusRd etc. for "Invalid" state)

Image Source: Culler, David (1997). Parallel Computer Architecture, p.286

MSI Protocol

t	Local Request	P1	P2	P3	Gen. Bus Request	Data Supplier
0	Initially	-	-	-	-	-
1	R1	S	-	-	BusRd	Memory
2	W1	M	-	-	BusUpgr	-
3	R3	S	-	S	BusRd	P1's cache (flush)

MESI Protocol

t	Local Request	P1	P2	P3	Gen. Bus Request	Data Supplier
0	Initially	-	-	-	-	-
1	R1	E	-	-	BusRd	Memory
2	W1	M	-	-	-	-
3	R3	S	-	S	BusRd	P1's cache (flush)

Exercise Tasks

1. Scaling behavior
2. Cache coherence
3. **Key figures of networks**
4. Task dependency graphs

- **Given: fully connected network (with N nodes)**

a) Number of edges?

→ Standard handshaking problem: each node is connected with all others.

Count each edge only once: $number\ of\ edges = \frac{N(N-1)}{2}$

b) Edge connectivity?

→ You need to remove all edges to one node to split the network, thus edge connectivity is N-1

c) Diameter? → 1

d) Bisection bandwidth?

- → After bisecting the fully connected network with N nodes you have two fully connected networks with N/2 nodes in case of even N
- → **Case 1:** N even: Two equal halves of size $\frac{N}{2}$

$$B_b = \frac{N(N-1)}{2} - 2\frac{\frac{N}{2}\left(\frac{N}{2}-1\right)}{2} = \frac{N^2}{2} - \frac{N}{2} - \frac{N^2}{4} + \frac{N}{2} = \frac{N^2}{4}$$

- → If N is odd: One network with $\frac{N+1}{2}$ nodes and one network with $\frac{N-1}{2}$ nodes
- → **Case 2:** N odd: Network with $\frac{N+1}{2}$ nodes and network with $\frac{N-1}{2}$ nodes

$$B_b = \frac{N(N-1)}{2} - \frac{\frac{N+1}{2}\left(\frac{N+1}{2}-1\right)}{2} - \frac{\frac{N-1}{2}\left(\frac{N-1}{2}-1\right)}{2}$$

$$= \frac{1}{2}(N(N-1) - \frac{(N+1)(N-1)}{4} - \frac{N-1}{2}\left(\frac{N-1}{2}-1\right))$$

$$= \frac{1}{2}(N^2 - N - \frac{N^2-1}{4} - \frac{N^2-2N+1}{4} + \frac{N-1}{2})$$

$$= \frac{1}{2}\left(N^2 - \frac{N^2}{4} - \frac{N^2}{4} - N + \frac{N}{2} + \frac{N}{2} + \frac{1}{4} - \frac{1}{4} - \frac{1}{2}\right) = \frac{1}{2}\left(\frac{N^2}{2} - \frac{1}{2}\right) = \frac{N^2}{4} - \frac{1}{4}$$

3.1 Fully connected network

- **Combining both cases: The bisection bandwidth for a fully connected network with N nodes and bandwidth 1 on each link is $\mathbf{B_b} = \lfloor\frac{N^2}{4}\rfloor$.**

- **Sketch a LogP model diagram for the global reduction-to-all communication operation**

  → Given: P processes = 8, vectors of size n, binary reduce operation, computation time $T_c(n) = 1$, network transmission time $T_n(n) = 1$

  a) Cyclic reduction

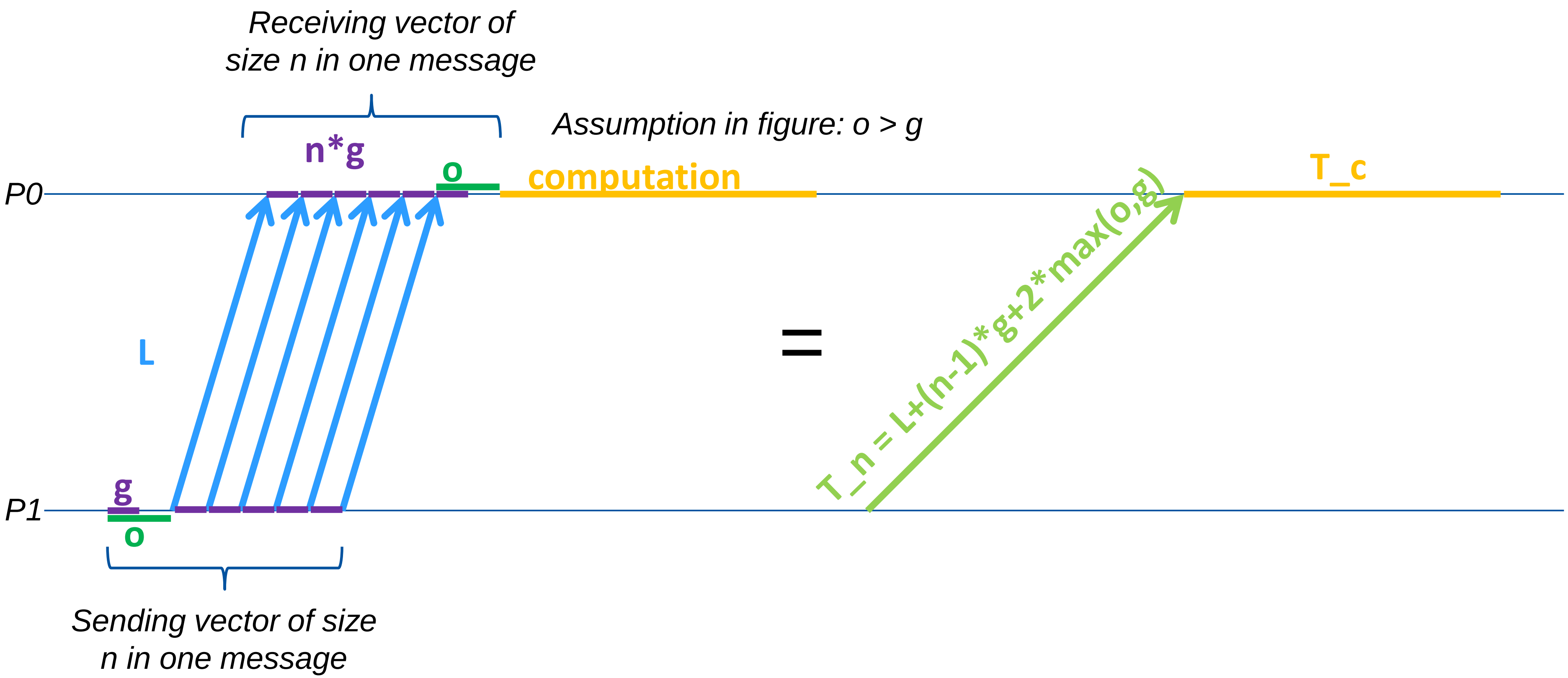
  b) Reduce-and-broadcast

     → Binary tree

     → Binomial tree

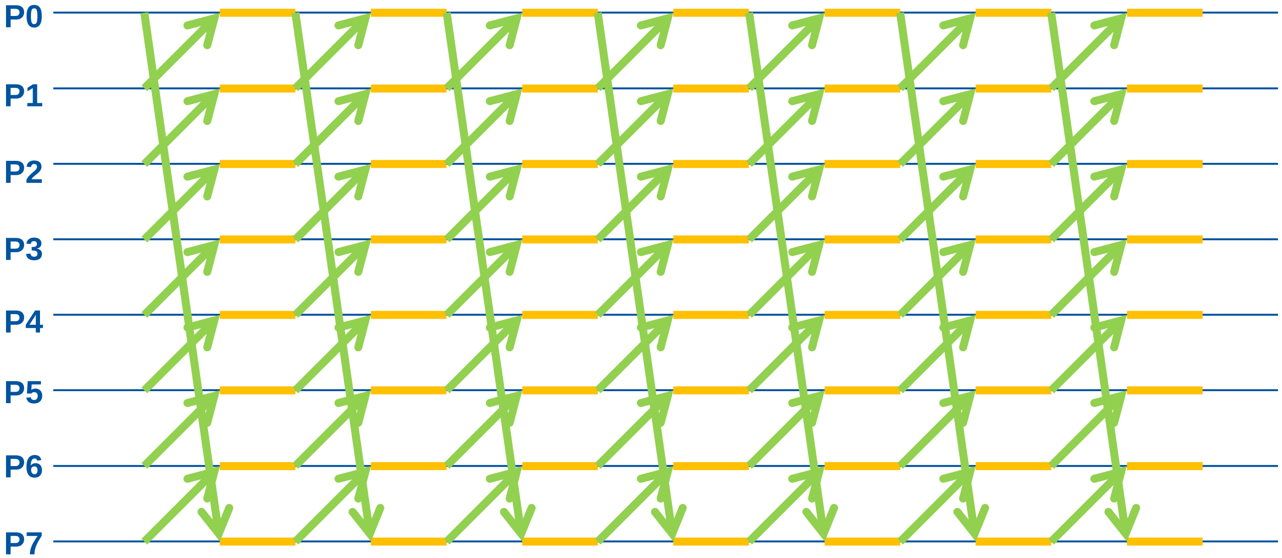
- **Compare the total runtime**

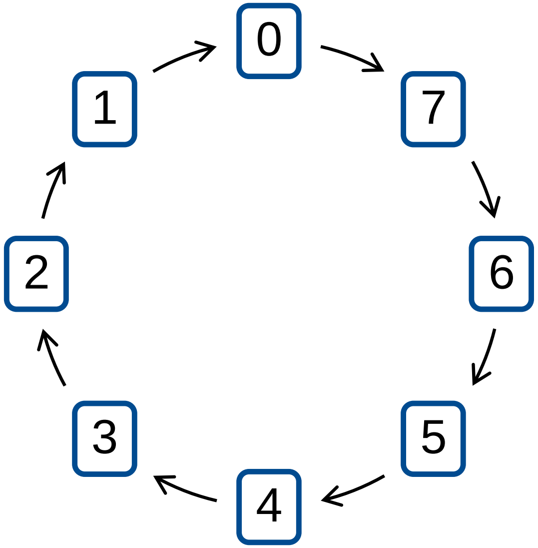
- **What potential effect has streaming (communicate calculated values) for large vectors?**



Assumptions on the following slides:
$T_c = 1$, $T_n = 1$

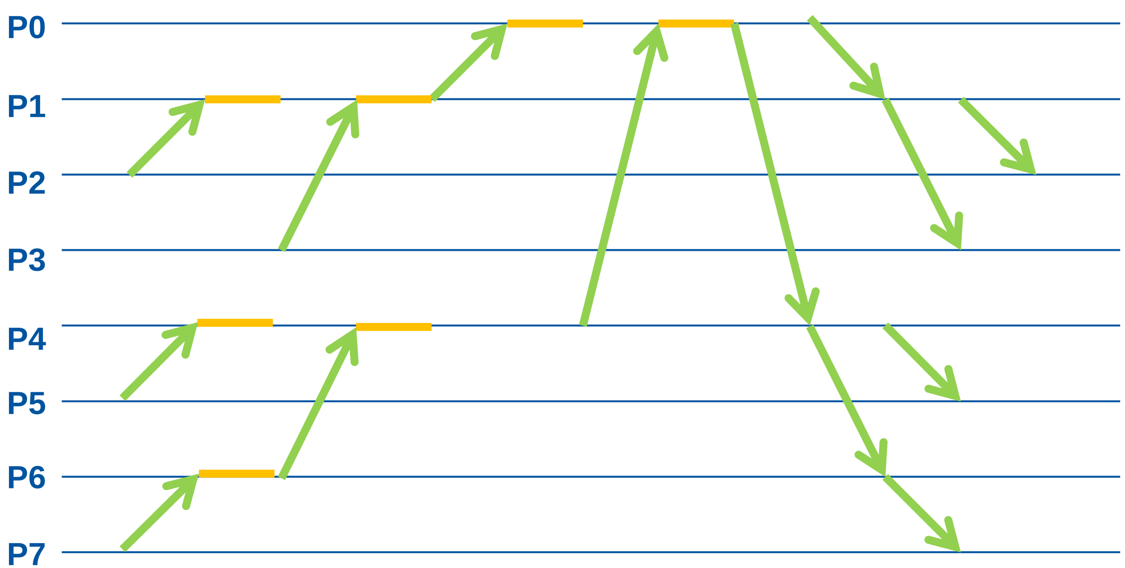
- **Cyclic reduction:** $(P-1)\,(T_c+T_n) = 7 \cdot 2 = 14$

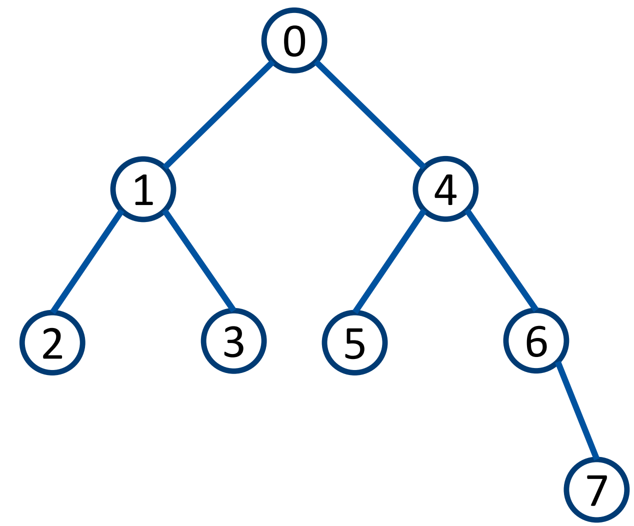




3. Key figures of networks
3.2 LogP Model

- **Binary tree:**
  - → $\mathbf{reduction} = 4\,(T_c + T_n) = 4 \cdot 2 = 8$
  - → $\mathbf{broadcast} = 4\,T_n = 4 \cdot 1 = 4$
  - → $\mathbf{reduce-to-all} = 8 + 4 = 12$

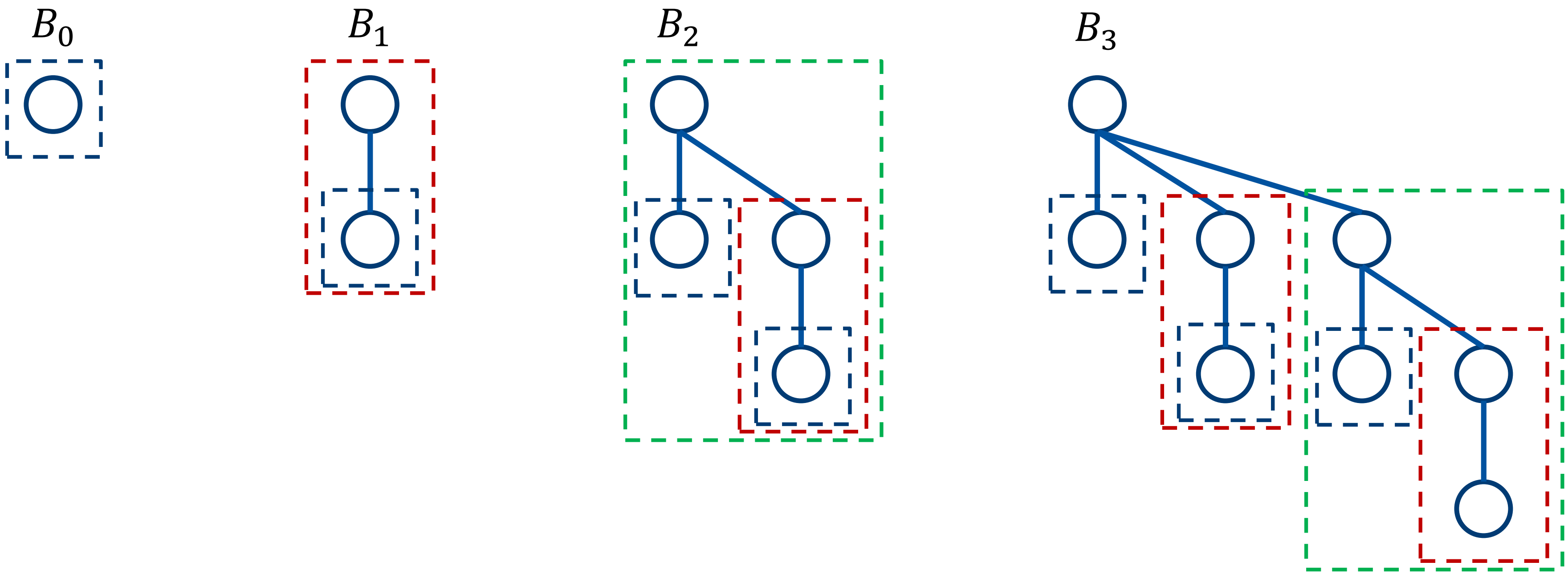




Assumption: communication & computation need roughly the same time

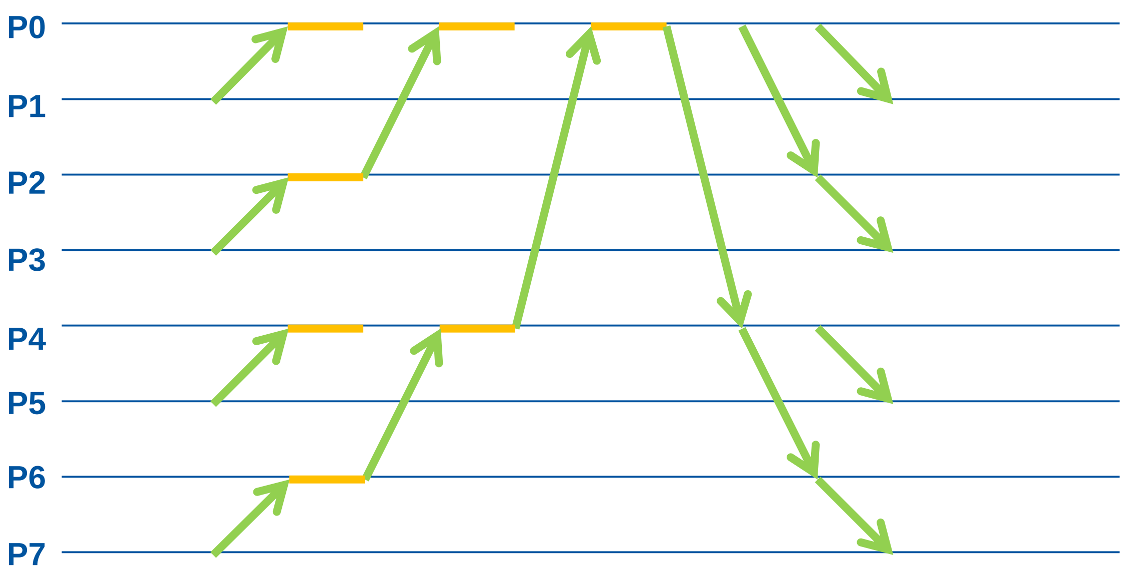
→ **Symmetric communication scheme**

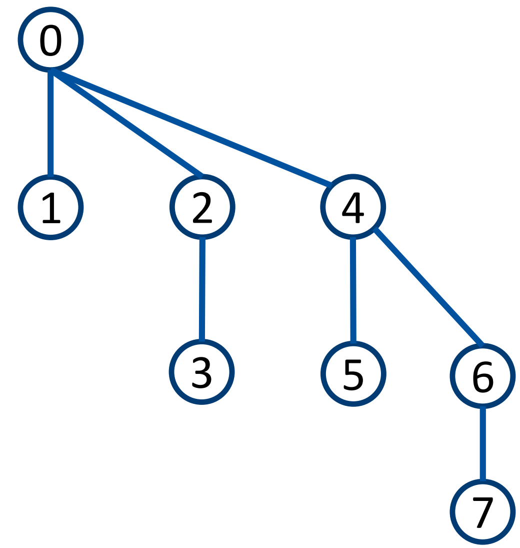
- **Binomial tree construction**
  - → B_0: Binomial tree of order 0 is a single node
  - → B_k: Binomial tree of order k has a root node with k children, namely the binomial trees of orders $0,1,2,\ldots,k-1,k-2$



3. Key figures of networks
3.2 LogP Model

- **Binomial tree:**
  - → $\mathbf{reduction} = \lceil \mathbf{ld}(P) \rceil \, (T_c + T_n) = 3 \cdot 2 = 6$
  - → $\boldsymbol{broadcast} = \lceil \boldsymbol{ld}(P) \rceil \, (T_n) = 3 \cdot 1 = 3$
  - → $\mathbf{reduce - to - all} = 6 + 3 = 9$

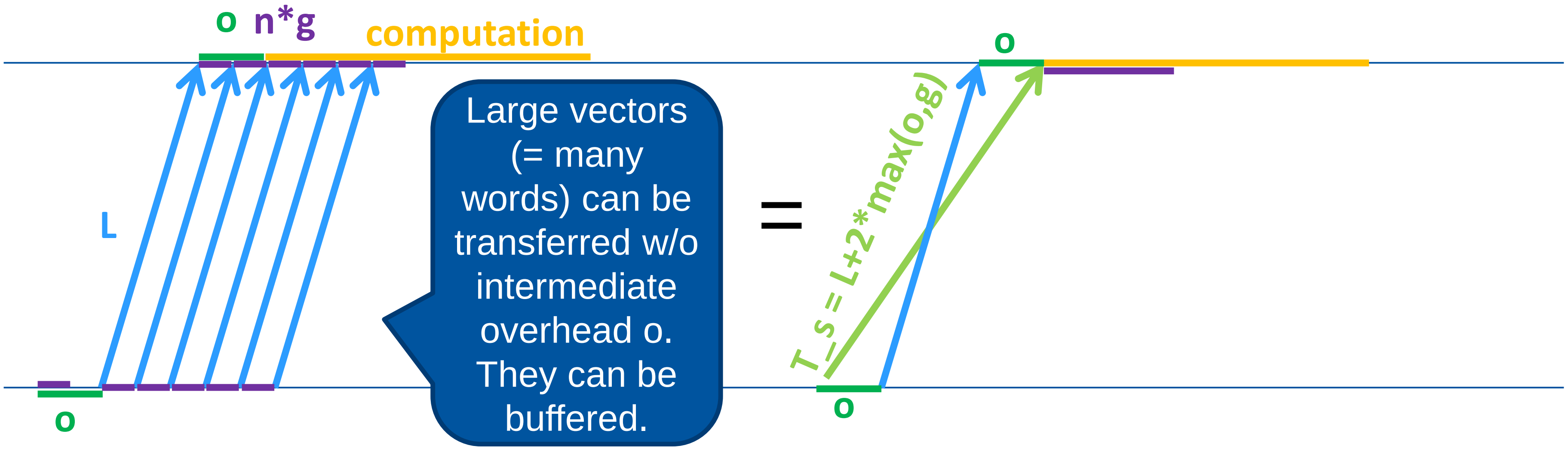




Assumption: communication & computation need roughly the same time

- → Better on broadcast and on reduction

- **c) Effect of streaming for large vectors**
- **For large transfers, LogP model is not the perfect model. Transmission is done by DMA controller, CPU is busy (*o*) just to start transfer.**



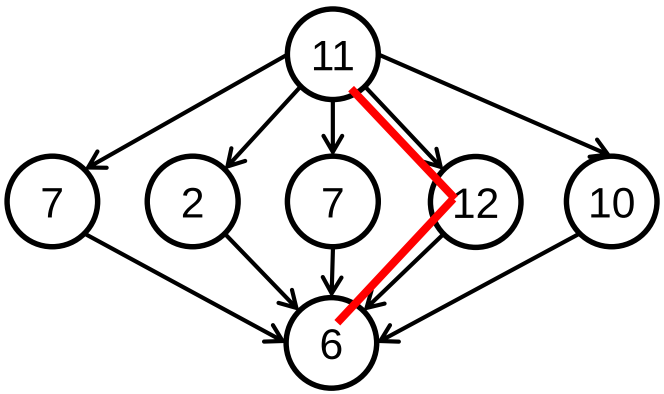
- **What effect does this have on the reduce-and-broadcast pattern?**

  → In the reduce steps, T_n can be replaced by T_s (due to overlapping computation and communication)

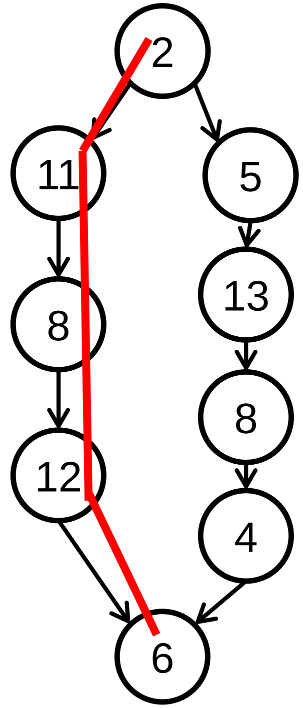
Exercise Tasks

1. Scaling behavior
2. Cache coherence
3. Key figures of networks
4. **Task dependency graphs**

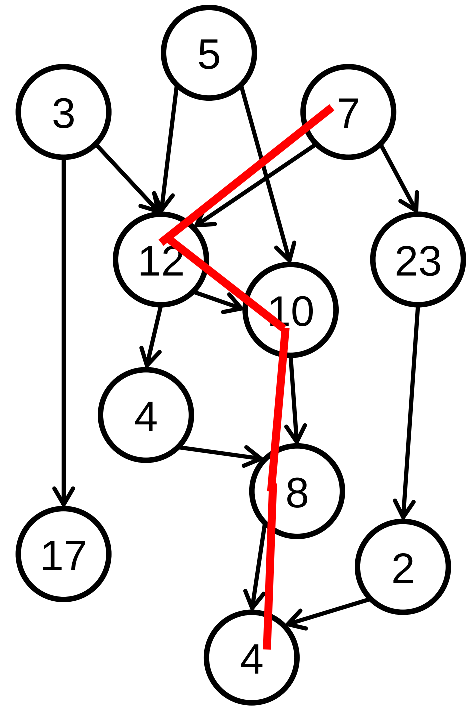
- **Determine: total work, critical path length, average concurrency**



Total work: 55
Critical path length: 29
Avg. concurrency: ~1.9



Total work: 69
Critical path length: 39
Avg. concurrency: ~1.8



Total work: 95
Critical path length: 41
Avg. concurrency: ~2.3