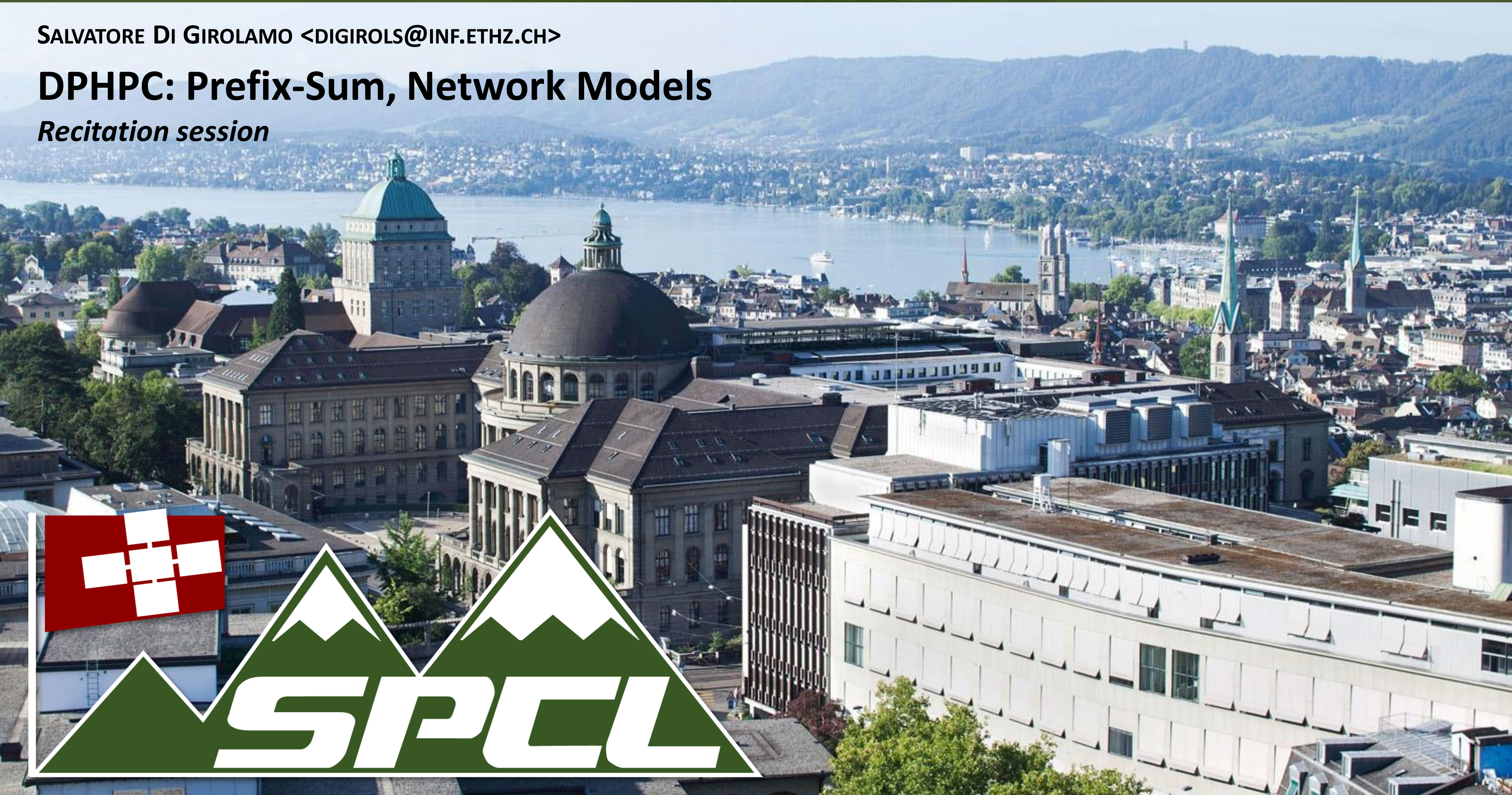
SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

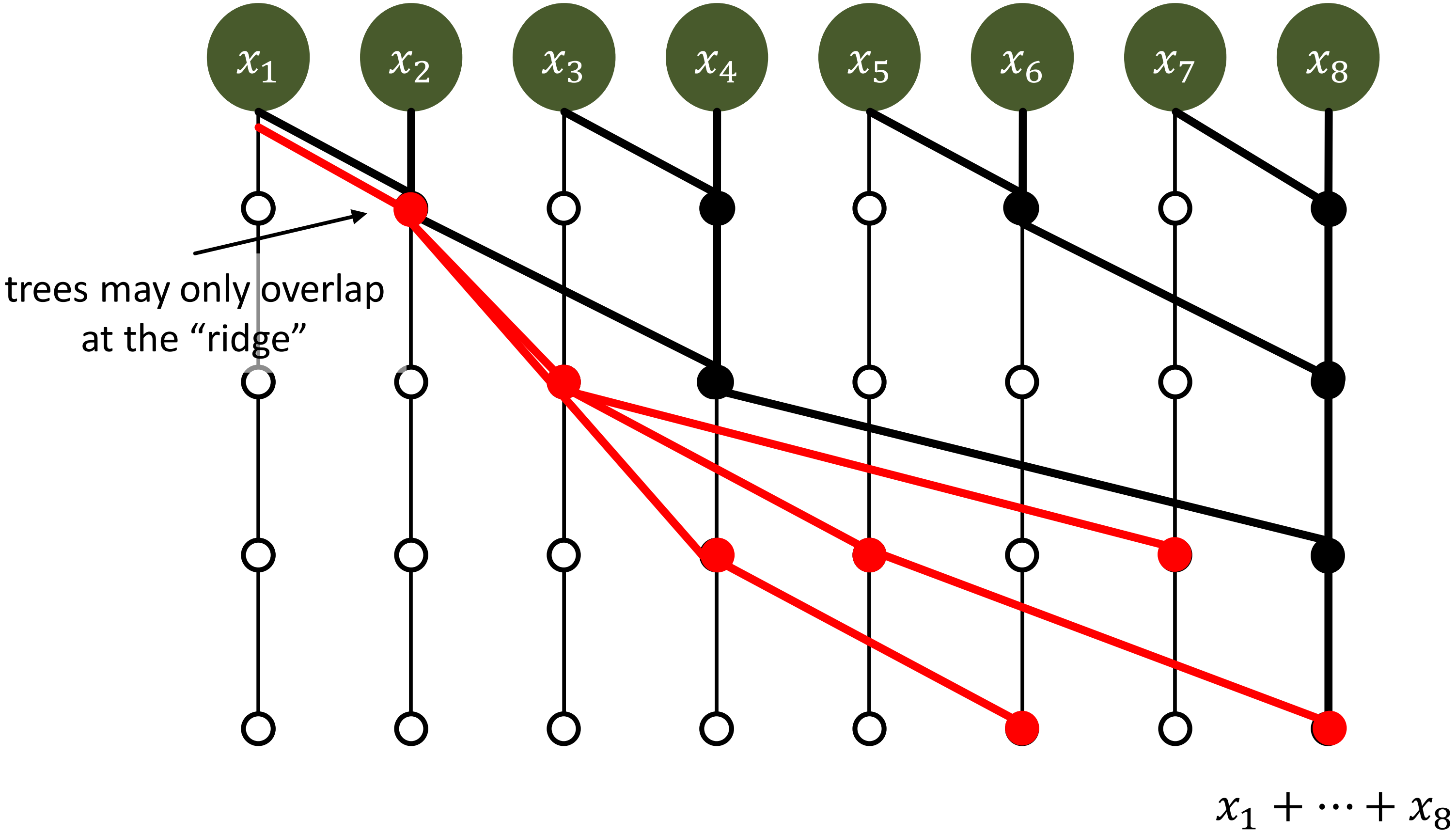
DPHPC: Prefix-Sum, Network Models

Recitation session



Recap: Work-depth tradeoff in parallel prefix sums

- **Obvious question: is there a depth- and work-optimal algorithm?**
  - This took years to settle! The answer is surprisingly: no
  - We know, for parallel prefix: $W + D \geq 2n - 2$



Output tree:

- leaves are all inputs, rooted at x_n
- binary due to binary operation
- $W = n - 1, D = D_o$

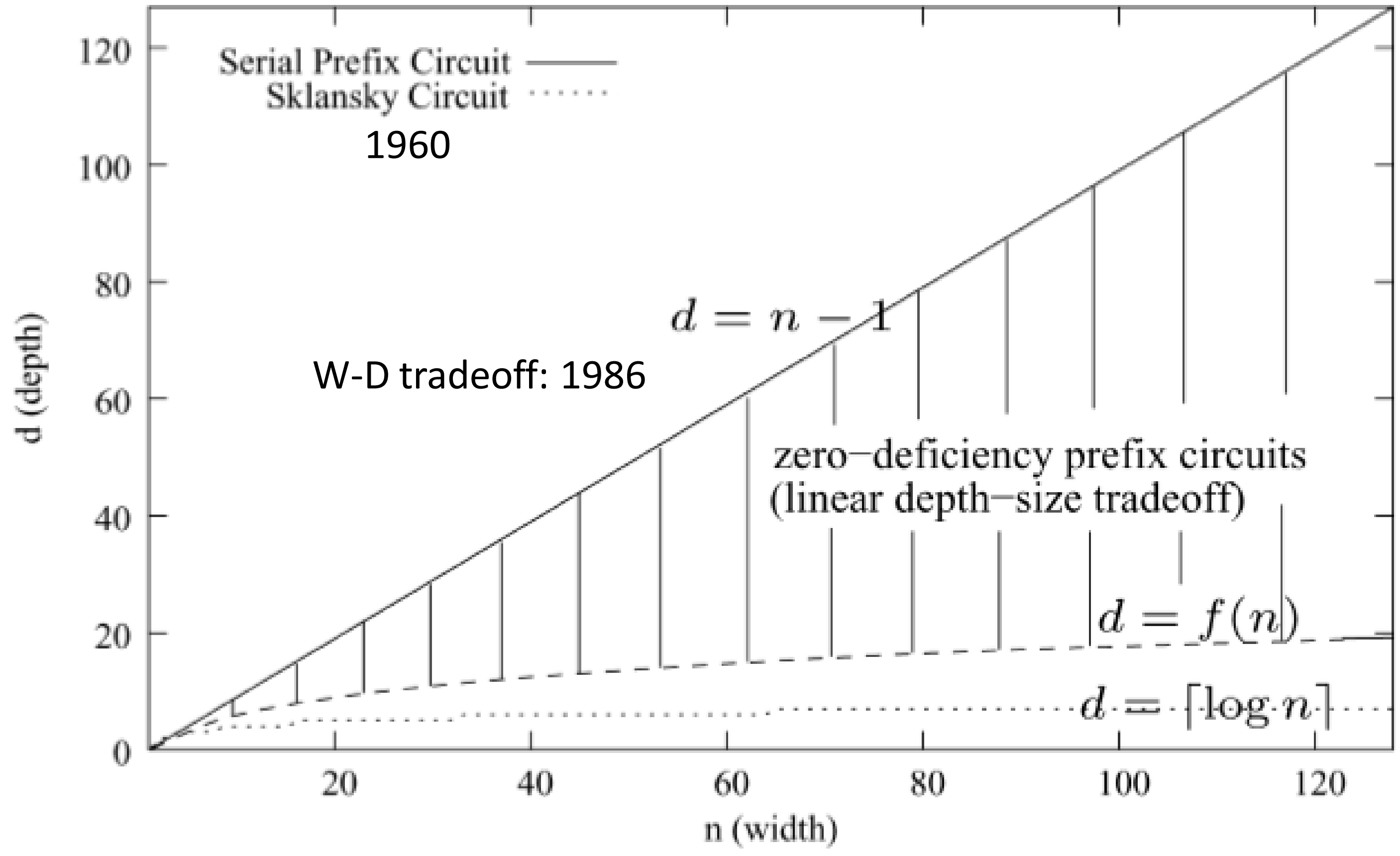
Input tree:

- rooted at x_1, leaves are all outputs
- not binary (simultaneous read)
- $W = n - 1$

Ridge can be at most D_o long!
Now add trees and subtract shared vertices:
$(n-1) + (n-1) - D_o = 2n - 2 - D_o \leq W$
q.e.d.

Work-Depth Tradeoffs and deficiency

“The deficiency of a prefix circuit c is defined as $\mathrm{def}(c) = W_c + D_c - (2n-2)$”



Latest 2006 result for zero-deficiency construction for $n > F(D+3) - 1$ ($f(n)$ is inverse)

From Zhu et al.: “Construction of Zero-Deficiency Parallel Prefix Circuits”, 2006

Prefix sums as magic bullet for other seemingly sequential algorithms

- **Any time a sequential chain can be modeled as function composition!**
  - Let $f_1, \ldots, f_n$ be an ordered set of functions and $f_0(x) = x$
  - Define ordered function compositions: $f_1(x); f_2(f_1(x)); \ldots ; f_n(\ldots f_1(x))$
  - If we can write function composition $g(x) = f_i(f_{i-1}(x))$ as $g = f_i \circ f_{i-1}$ then we can compute $\circ$ with a prefix sum!

    We saw an example with the adder (M_{ab} were our functions)

- **Example: linear recurrence $\boldsymbol{f_i(x) = a_i f_{i-1}(x) + b_i}$ with $\boldsymbol{f_0(x)}$=x**
  - Write as matrix form $f_i\binom{x}{1} = \begin{pmatrix} a_i & b_i \\ 0 & 1 \end{pmatrix} f_{i-1}\binom{x}{1}$
  - Function composition is now simple matrix multiplication!

    For example: $f_2\binom{x}{1} = \begin{pmatrix} a_2 & b_2 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} a_1 & b_1 \\ 0 & 1 \end{pmatrix} f_0\binom{x}{1} = \begin{pmatrix} a_1 a_2 & a_2 b_1 + b_2 \\ 0 & 1 \end{pmatrix}\binom{x}{1}$

Parallel Filter

Given an array, produce an array containing only the elements for which f(e) is true

```
input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]

f(e): true if e > 10

output [17, 11, 13, 19, 24]
```

Parallelizable?

https://wiki.rice.edu/confluence/download/attachments/4435861/comp322-s18-lec37-slides-v1.pdf?version=1&modificationDate=1523898025646&api=v2

Parallel Filter

```
input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]
```

- Parallel map to compute a bit-vector for true elements

```
bits  [1,  0, 0, 0,  1, 0,  1,  1, 0,  1]
```

- Parallel-prefix sum on the bit-vector

```
bitsum [1,  1, 1, 1,  2, 2,  3,  4, 4,  5]
```

- Parallel map to produce the output

```
output  = new array of size bitsum[n-1]
FORALL(i=0; i < input.length; i++){
  if(bits[i]==1)
    output[bitsum[i]-1] = input[i];
}
```

```
output [17, 11, 13, 19, 24]
```

https://wiki.rice.edu/confluence/download/attachments/4435861/comp322-s18-lec37-slides-v1.pdf?version=1&modificationDate=1523898025646&api=v2

Parallel Lexical Analysis and Parsing

INPUT	(	f	o	o	space	+	"	s	"	)

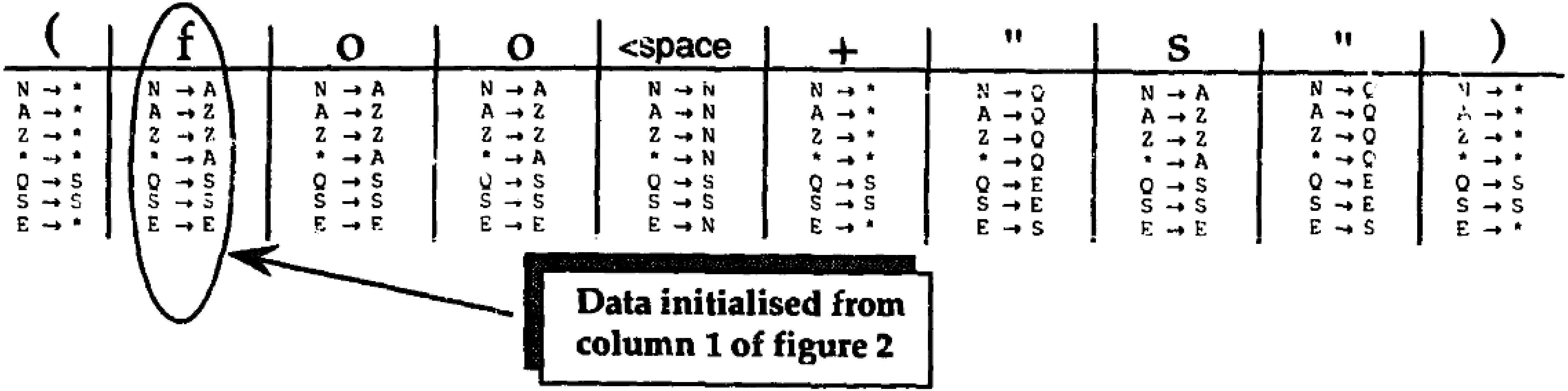
	letter	character token	Quote	White Space
N	A	*	Q	N
A	Z	*	Q	N
Z	Z	*	Q	N
*	A	*	Q	N
Q	S	S	E	S
S	S	S	E	N
E	E	*	S	N

Hill, Jonathan MD. "Parallel lexical analysis and parsing on the AMT distributed array processor." *Parallel computing* 18.6 (1992): 699-714.

Parallel Lexical Analysis and Parsing

- **Substitute each character with its associated transition (i.e., column of the FSM)**

	letter	character token	Quote	White Space
N	A	*	Q	N
A	Z	*	Q	N
Z	Z	*	Q	N
*	A	*	Q	N
Q	S	S	E	S
S	S	S	E	N
E	E	*	S	N



Parallel Lexical Analysis and Parsing

- **Define a composition function as a function that given two transitions $\alpha \rightarrow \beta$ and $\beta \rightarrow \delta$, produces $\alpha \rightarrow \delta$.**

(	f	o	o	<space	+	"	S	"	)
N → *	N → A	N → A	N → A	N → N	N → *	N → Q	N → A	N → Q	N → *
A → *	A → Z	A → Z	A → Z	A → N	A → *	A → Q	A → Z	A → Q	A → *
Z → *	Z → Z	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → Z	Z → Q	Z → *
* → *	* → A	* → A	* → A	* → N	* → *	* → Q	* → A	* → Q	* → *
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → S	Q → E	Q → S
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → S	S → E	S → S
E → *	E → E	E → E	E → E	E → N	E → *	E → S	E → E	E → S	E → *

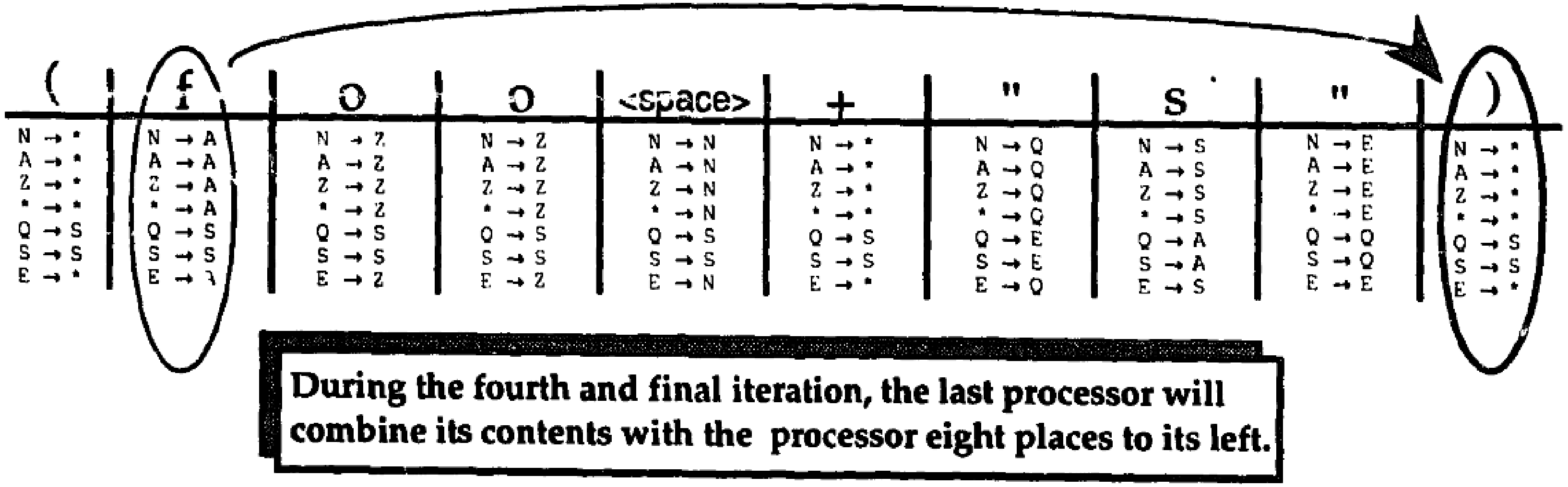
Data initialised from column 1 of figure 2

(	f	o	o	<space	+	"	S	"	)
N → *	N → A	N → Z	N → Z	N → N	N → *	N → Q	N → S	N → Q	N → S
A → *	A → A	A → Z	A → Z	A → N	A → *	A → Q	A → S	A → Q	A → S
Z → *	Z → A	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → S	Z → Q	Z → S
* → *	* → A	* → Z	* → Z	* → N	* → *	* → Q	* → S	* → Q	* → S
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → E	Q → E	Q → *
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → E	S → E	S → *
E → *	E → A	E → Z	E → Z	E → N	E → *	E → O	E → E	E → S	E → S

The left and right processing elements compose their state transition tables and the result is stored in the right of the two elements. This result signifies that the machine would be in state A or S after reading the previous and current characters (i.e "(f")

Parallel Lexical Analysis and Parsing

- **Apply prefix sum.**
- **At the end, the state of each processor is equivalent to the that produced by a finite state machine in all possible states after reading the current and all the previous characters.**

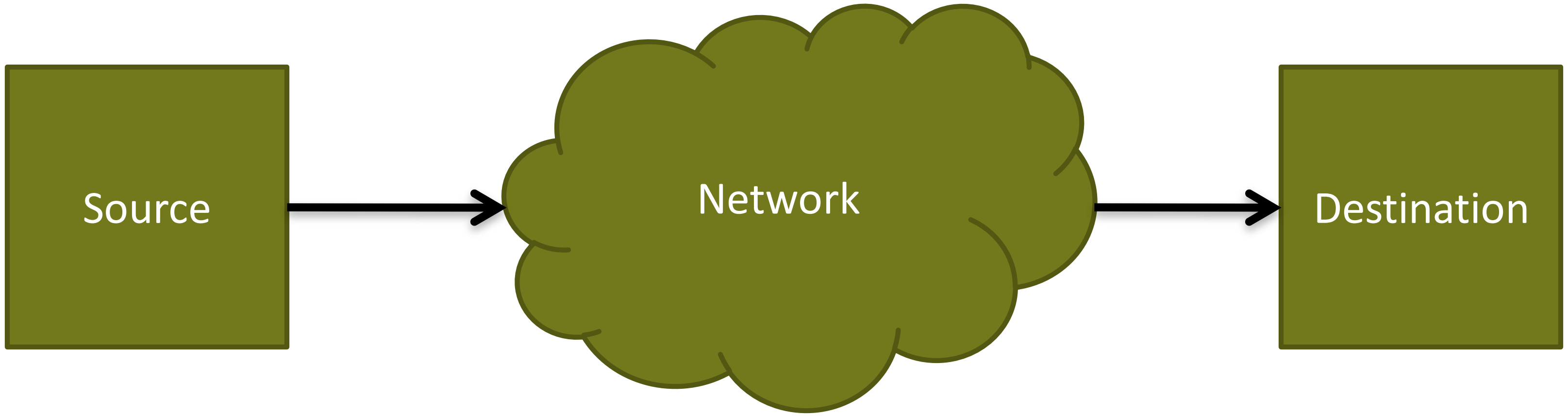


INPUT	(	f	o	o	space	+	"	s	"	)
Result State	*	A	Z	Z	N	*	Q	S	E	*
Comment	Single char. token	'A' denotes the start of a identifier token , and Z' corresponds to the continuation of that token			ignore	Single char token	'Q' and 'E' denote the start & end of a quotes token, & S denotes the sentence within the quotes			Single char token

Hill, Jonathan MD. "Parallel lexical analysis and parsing on the AMT distributed array processor." *Parallel computing* 18.6 (1992): 699-714.

Distributed networking basics

- **Familiar (non-HPC) network: Internet TCP/IP**
  - Common model:



- **Class Question: What parameters are needed to model the performance (including pipelining)?**
  - Latency, Bandwidth, Injection Rate, Host Overhead
  - What network models do you know and what do they model?

Remember: A Simple Model for Communication

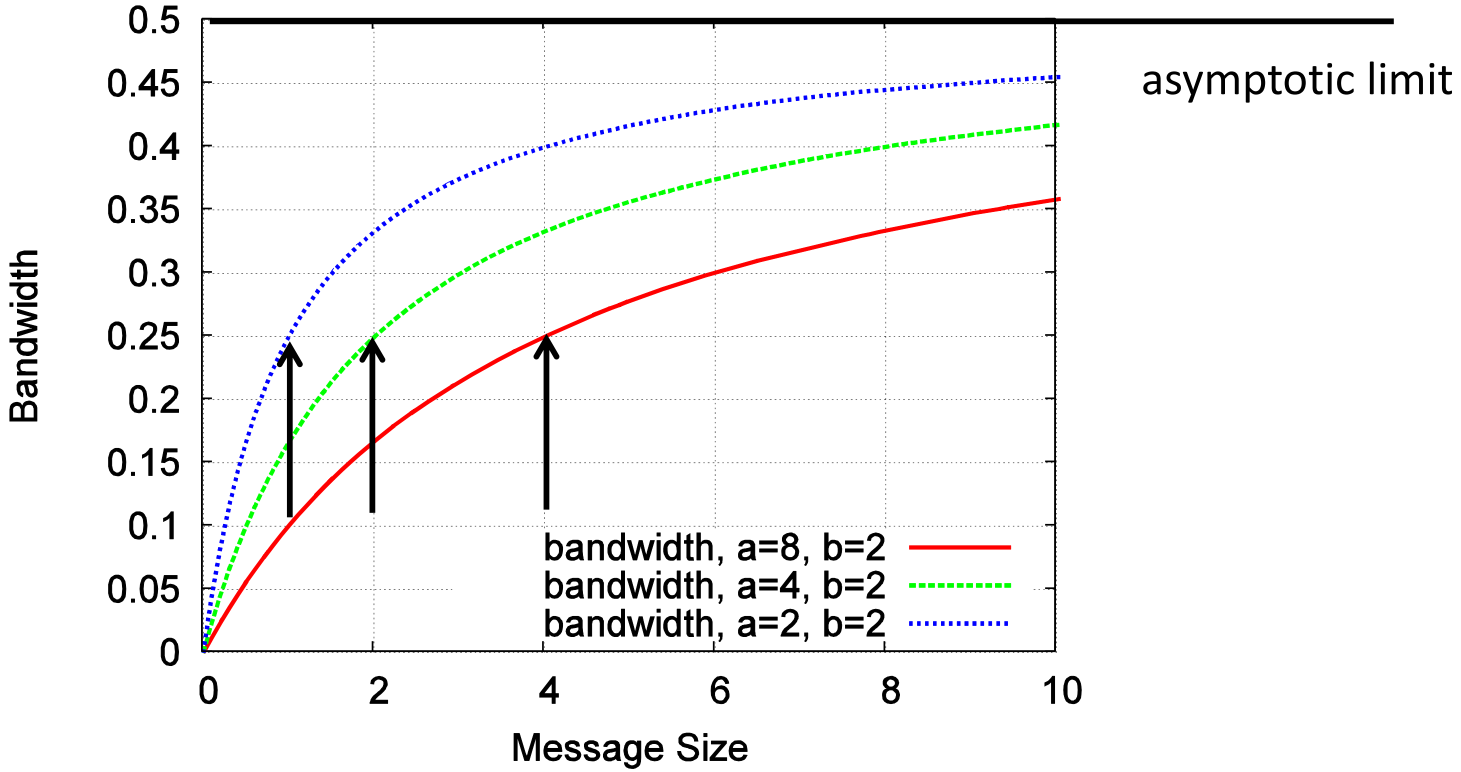
- **Transfer time** $T(s) = \alpha + \beta s$
  - α = startup time (latency)
  - β = cost per byte (bandwidth=1/β)

- **As s increases, bandwidth approaches** $1/\beta$ **asymptotically**
  - Convergence rate depends on α
  - $s_{\frac{1}{2}} = \alpha/\beta$

- **Assuming no pipelining (new messages can only be issued from a process after all arrived)**

Bandwidth vs. Latency

- $s_{\frac{1}{2}} = \alpha/\beta$ **is often used to distinguish bandwidth- and latency-bound messages**
  - $s_{\frac{1}{2}}$ is in the order of kilobytes on real systems



Quick Example

- **Simplest linear broadcast**
  - One process has a data item to be distributed to all processes

- **Linearly broadcasting s bytes among P processes:**
  - $T(s) = (P-1) \cdot (\alpha + \beta s) = O(P)$

- **Class question: Do you know a faster method to accomplish the same?**

k-ary Tree Broadcast

- **Origin process is the root of the tree, passes messages to k neighbors which pass them on**
  - k=2 -> binary tree

- **Class Question: What is the broadcast time in the simple latency/bandwidth model?**
  - $T(s) \approx \lceil \log_k P \rceil \cdot k(\alpha + \beta s)$ (for fixed k)

- **Class Question: What is the optimal k?**

  - $0 = \frac{k \ln P}{\ln k} \frac{d}{dk} = \frac{\ln P \ln k - \ln P}{\ln^2 k} \rightarrow k = e = 2.71 \ldots$

  - Independent of $P, \alpha, \beta s$? Really?

Faster Trees?

- **Class Question: Can we broadcast faster than in a ternary tree?**
  - Yes because each respective root is idle after sending three messages!
  - Those roots could keep sending!
  - Result is a k-nomial tree

    For k=2, it's a binomial tree

- **Class Question: What about the runtime?**
  - $T(s) = \lceil log_k(P) \rceil \cdot (k-1) \cdot (\alpha + \beta \cdot s) = \mathcal{O}(log(P))$

- **Class Question: What is the optimal k here?**
  - T(s) d/dk is monotonically increasing for k>1, thus $k_{opt}=2$

- **Class Question: Can we broadcast faster than in a k-nomial tree?**
  - $\mathcal{O}(log(P))$ is asymptotically optimal for s=1!
  - But what about large s?

Very Large Message Broadcast

- **Extreme case (P small, s large): simple pipeline**
  - Split message into segments of size z
  - Send segments from PE i to PE i+1
- **Class Question: What is the runtime?**
  - T(s) = (P-2+s/z)(α + βz)
- **Compare 2-nomial tree with simple pipeline for α=10, β=1, P=4, s=10^6, and z=10^5**
  - 2,000,020 vs. 1,200,120
- **Class Question: Can we do better for given α, β, P, s?**
  - Derive by z $z_{opt} = \sqrt{\frac{s\alpha}{(P-2)\beta}}$
- **What is the time for simple pipeline for α=10, β=1, P=4, s=10^6, z_{opt}?**
  - 1,008,964

Lower Bounds

- **Class Question: What is a simple lower bound on the broadcast time?**
  - $T_{BC} \geq \min\{\lceil \log_2(P) \rceil \alpha, s\beta\}$
- **How close are the binomial tree for small messages and the pipeline for large messages (approximately)?**
  - Bin. tree is a factor of $\log_2$(P) slower in bandwidth
  - Pipeline is a factor of P/$\log_2$(P) slower in latency
- **Class Question: What can we do for intermediate message sizes?**
  - Combine pipeline and tree → pipelined tree
- **Class Question: What is the runtime of the pipelined binary tree algorithm?**
  - $T \approx \left(\frac{s}{z} + \lceil \log_2 P \rceil - 2\right) \cdot 2 \cdot (\alpha + z\beta)$
- **Class Question: What is the optimal z?**
  - $z_{opt} = \sqrt{\frac{\alpha s}{\beta(\lceil \log_2 P \rceil - 2)}}$

Towards an Optimal Algorithm

- **What is the complexity of the pipelined tree with z_{opt} for small s, large P and for large s, constant P?**
  - Small messages, large P: s=1; z=1 (s≤z), will give O(log P)
  - Large messages, constant P: assume α, β, P constant, will give asymptotically O(sβ)

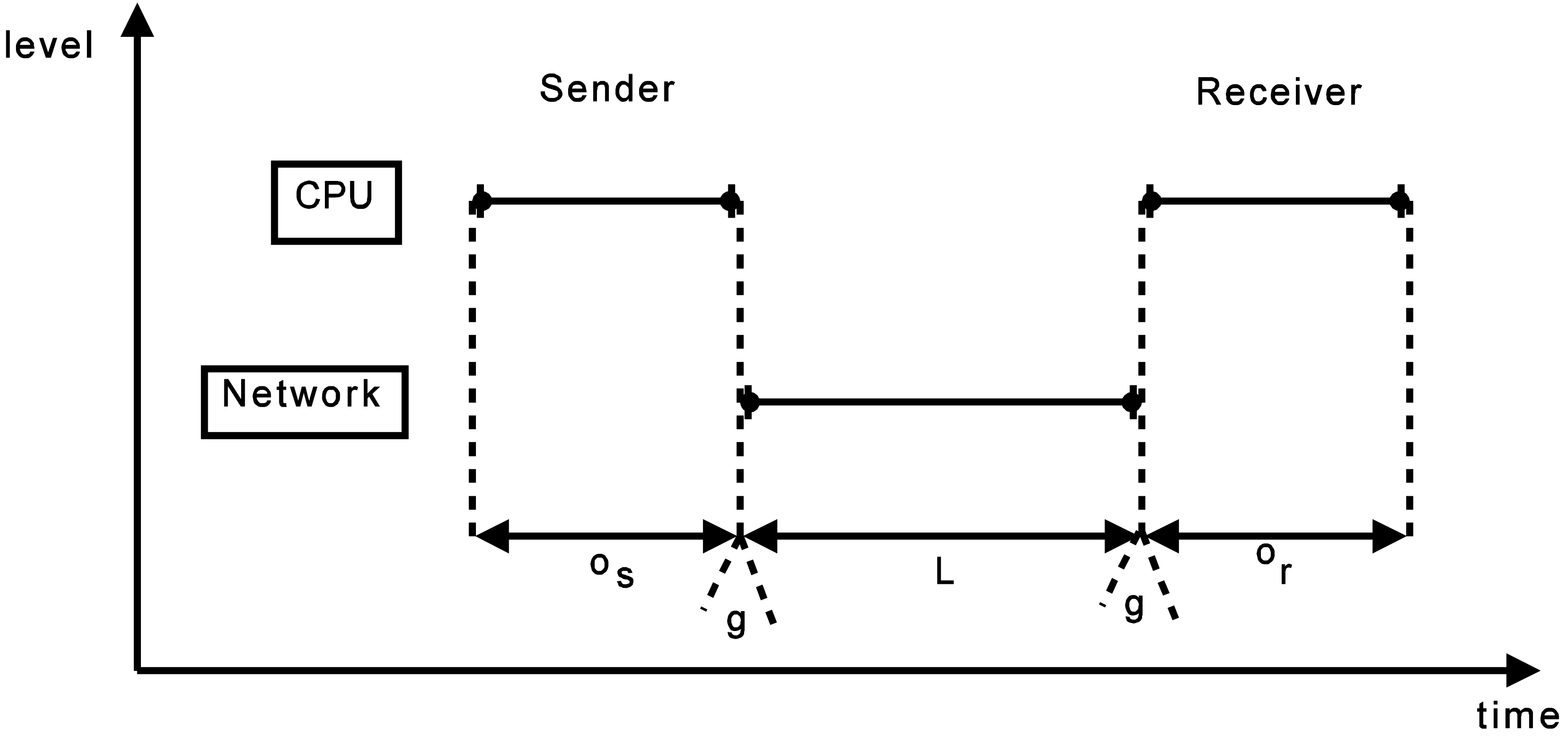
    Asymptotically optimal for large P and s but bandwidth is off by a factor of 2! Why?
- **Bandwidth-optimal algorithms exist, e.g., Sanders et al. *"Full Bandwidth Broadcast, Reduction and Scan with Only Two Trees".* 2007**
  - Intuition: in binomial tree, all leaves (P/2) only receive data and never send → wasted bandwidth
  - Send along two simultaneous binary trees where the leafs of one tree are inner nodes of the other
  - Construction needs to avoid endpoint congestion (makes it complex)

    Can be improved with linear programming and topology awareness

The LogP Model

- **Defined by four parameters:**
  - L: an upper bound on the latency, or delay, incurred in communicating a message containing a word (or small number of words) from its source module to its target module.
  - o: the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations.
  - g: the gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor. The reciprocal of g corresponds to the available per-processor communication bandwidth.
  - P: the number of processor/memory modules. We assume unit time for local operations and call it a cycle.

The LogP Model



Simple Examples

- **Sending a single message**
  - $T = 2o+L$
- **Ping-Pong Round-Trip**
  - $T_{RTT} = 4o+2L$
- **Transmitting n messages**
  - $T(n) = L+(n-1)*\max(g, o) + 2o$

Simplifications

- **o is bigger than g on some machines**
  - g can be ignored (eliminates max() terms)
  - be careful with multicore!
- **Offloading networks might have very low o**
  - Can be ignored (not yet but hopefully soon)
- **L might be ignored for long message streams**
  - If they are pipelined
- **Account g also for the first message**
  - Eliminates “-1”

Benefits over Latency/Bandwidth Model

- **Models pipelining**
  - L/g messages can be "in flight"
  - Captures state of the art (cf. TCP windows)
- **Models computation/communication overlap**
  - Asynchronous algorithms
- **Models endpoint congestion/overload**
  - Benefits balanced algorithms

Example: Broadcasts

- **Class Question: What is the LogP running time for a linear broadcast of a single packet?**
  - $T_{lin} = L + (P-2) * \max(o,g) + 2o$
- **Class Question: Approximate the LogP runtime for a binary-tree broadcast of a single packet?**
  - $T_{bin} \leq \log_2 P * (L + \max(o,g) + 2o)$
- **Class Question: Approximate the LogP runtime for an k-ary-tree broadcast of a single packet?**
  - $T_{k\text{-}n} \leq \log_k P * (L + (k-1)\max(o,g) + 2o)$

Example: Broadcasts

- **Class Question: Approximate the LogP runtime for a binomial tree broadcast of a single packet (assume L > g!)?**
  - $T_{bin} \leq \log_2 P * (L + 2o)$
- **Class Question: Approximate the LogP runtime for a k-nomial tree broadcast of a single packet?**
  - $T_{k\text{-}n} \leq \log_k P * (L + (k-2)\max(o,g) + 2o)$
- **Class Question: What is the optimal k (assume o>g)?**
  - Derive by k: $0 = o * \ln(k_{opt}) - L/k_{opt} + o$ (solve numerically)

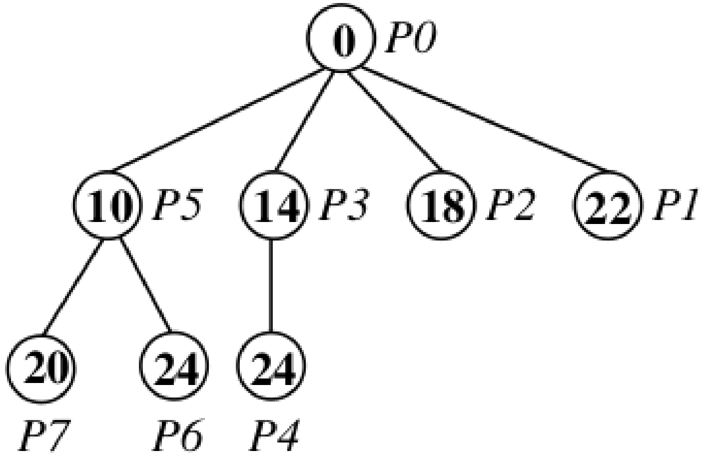
    For larger L, k grows and for larger o, k shrinks
  - Models pipelining capability better than simple model!

Example: Broadcasts

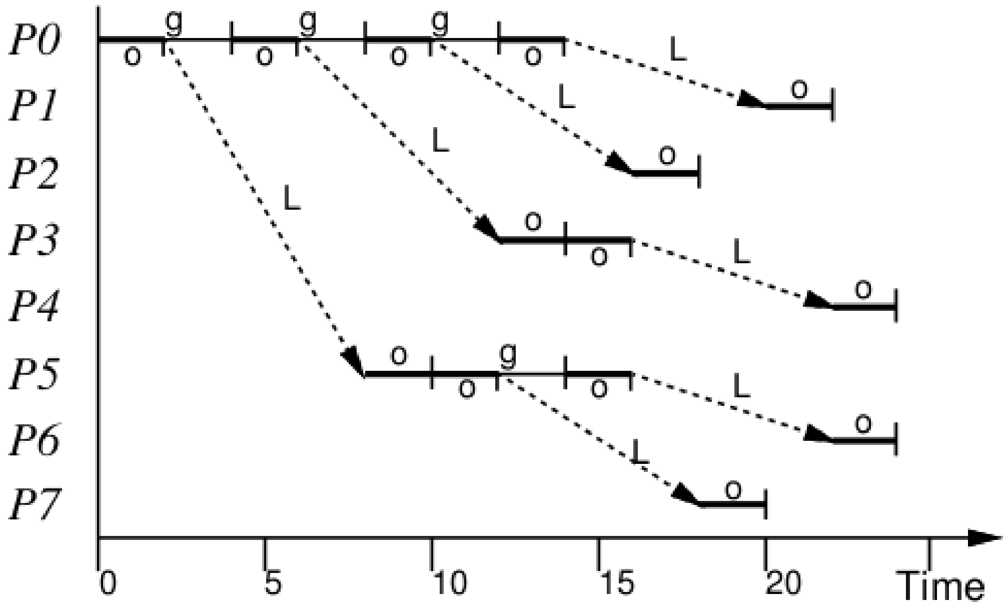
- **Class Question: Can we do better than k_{opt}-ary binomial broadcast?**
  - Problem: fixed k in all stages might not be optimal
  - We can construct a schedule for the optimal broadcast in practical settings
  - First proposed by Karp et al. in “Optimal Broadcast and Summation in the LogP Model”

Example: Optimal Broadcast

- **Broadcast to P-1 processes**
  - Each process who received the value sends it on; each process receives exactly once



P=8, L=6, g=4, o=2



Optimal Broadcast Runtime

- **This determines the maximum number of PEs (P(t)) that can be reached in time t**
- **P(t) can be computed with a generalized Fibonacci recurrence (assuming o>g):**

$$P(t) = \begin{cases} 1 : & t < 2o + L \\ P(t-o) + P(t-L-2o) : & \text{otherwise.} \end{cases} \quad (1)$$

- **Which can be bounded by (see [1]):** $2^{\left\lfloor \frac{t}{L+2o} \right\rfloor} \leq P(t) \leq 2^{\left\lfloor \frac{t}{o} \right\rfloor}$
  - A closed solution is an interesting open problem!

[1]: Hoefler et al.: "Scalable Communication Protocols for Dynamic Sparse Data Exchange" (Lemma 1)