University of Nijmegen

Splitters

Objects For On-Line Partitioning

Jaap-Henk Hoepman
Department of Computer Science
University of Nijmegen, the Netherlands
jhh@cs.kun.nl
www.cs.kun.nl/~jhh

University of Nijmegen

Contents

University of Nijmegen

What is a splitter?

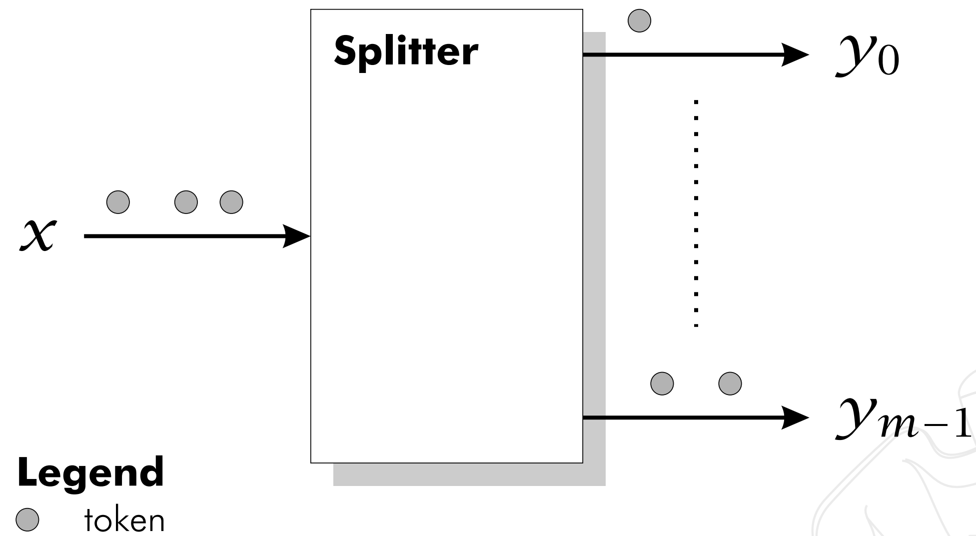
A concurrent asynchronous non-blocking object that can partition a collection of contending tokens into smaller groups with certain properties.

Used many times before in the literature:

- mutual exclusion [Lam87],
- renaming [MA95, AM94, BGHM95], and
- resource allocation [AHS94].

But never studied independently (except for counting networks [AHS94]).

Splitters



- 1 input x, m outputs y_i.
- shared among n processes.
- tokens enter and release.
- one-shot vs. long-lived.
- Token states *idle*, *entering*, *assigned*, and *releasing*.

Research questions

- What does it require to implement a certain splitter?
- How can splitters be combined to implement other splitters?

But first. . .

How do we define splitters?

Contention (1)

For input or output z of S at time t:

point contention $\eth^t z$: the number of tokens at z at time t.

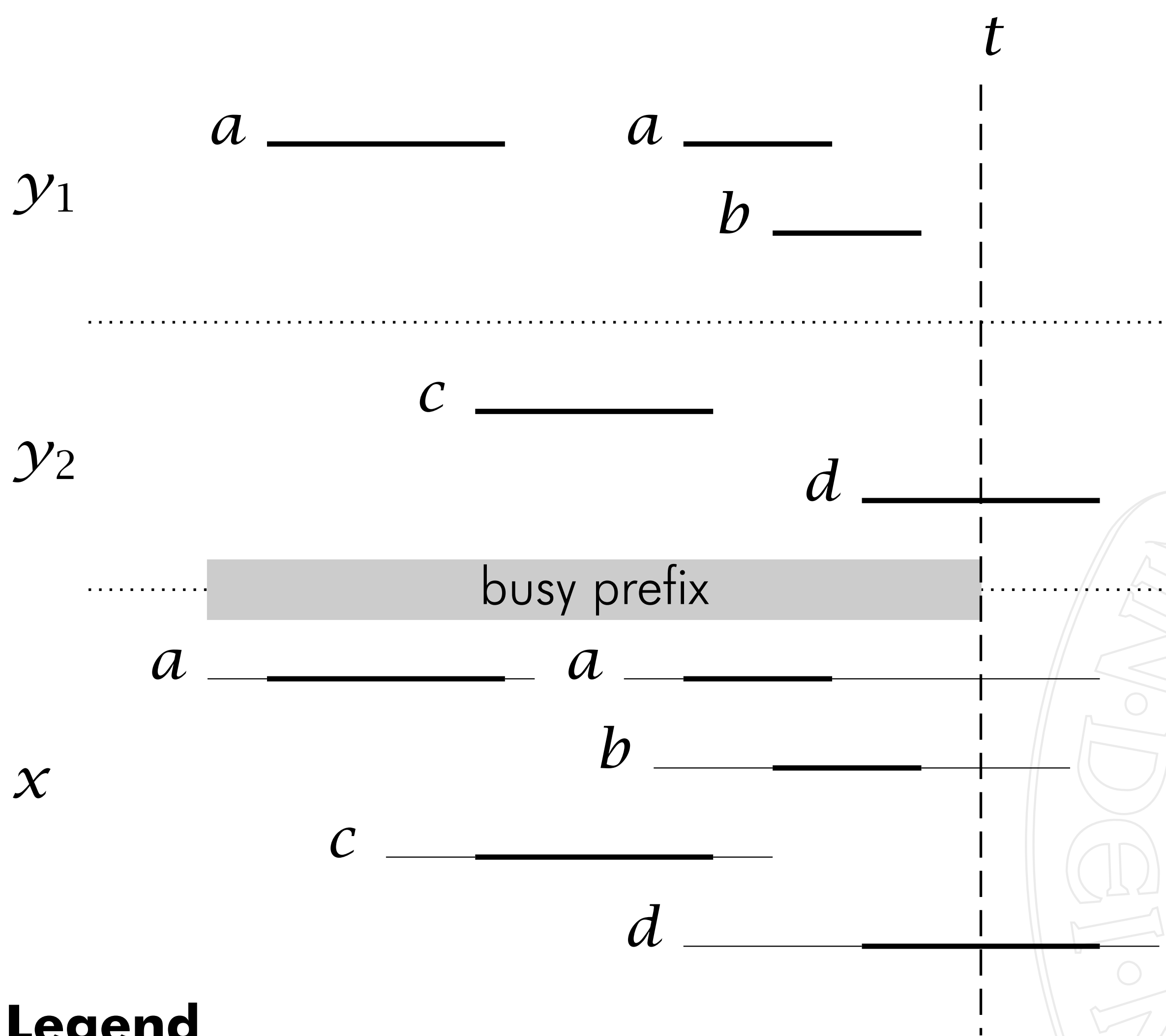
maximal point contention $\delta^t z$: the maximal number of tokens at z at any time t' within the busy prefix of S at t.

interval contention $\Delta^t z$: the total number of *different* tokens (i.e., not counting doubles) at z in the busy prefix of S at t.

total contention $\nabla^t z$: the total number of tokens (counting doubles) at z in the busy prefix of S at t.

University of Nijmegen

Contention (2)

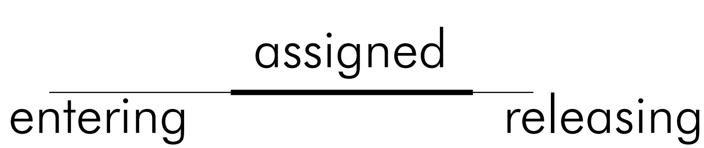


	$\eth^t$	δ^t	Δ^t	∇^t
y_1	0	2	2	3

	$\eth^t$	δ^t	Δ^t	∇^t
y_2	1	1	2	2

	$\eth^t$	δ^t	Δ^t	∇^t
x	3	4	4	5

Legend



Defining splitters

Invariant $\mathsf{Inv}(S)$

- Predicate over the states σ of S
  - *using only input contention dx and output contentions dy_i, where $d \in \{\eth, \delta, \Delta, \nabla\}$.*
- $\sigma \models \mathsf{Inv}(S)$ must hold for all states.

($\sigma \models P$ if predicate P holds in state σ)

Properties

For any state σ of splitter S with m outputs, and input or output z:

- $\eth z, \delta z, \Delta z, \nabla z \geq 0$,
- $\eth z \leq \delta z \leq \Delta z \leq \nabla z$ (equality for one-shot).
- $\sum_{i=1}^{m} \eth y_i \leq \eth x$ and $\sum_{i=1}^{m} \nabla y_i \leq \nabla x$ (equality in the steady state),

Axioms

Axiom 1 *Let σ be the state of splitter S with all tokens idle. Then $\sigma \models \mathsf{Inv}(S)$.*

Axiom 2 *For all states σ of a splitter S, if $\sigma \models \mathsf{Inv}(S)$ and for some token t we have $\sigma(t) = \to\!\odot$, then there is an i with $1 \leq i \leq m$ such that $\sigma(t) : i \models \mathsf{Inv}(S)$.*

For long-lived splitters only:

Axiom 3 *For all states σ of a splitter S, if $\sigma \models \mathsf{Inv}(S)$ and for some token t we have $\sigma(t) = i$ with $1 \leq i \leq m$, then $\sigma(t) : \odot\!\to \models \mathsf{Inv}(S)$.*

Axiom 4 *For all states σ of a splitter S, if $\sigma \models \mathsf{Inv}(S)$ and for some token t we have $\sigma(t) = \odot\!\to$ then $\sigma(t) : \perp \models \mathsf{Inv}(S)$.*

University of Nijmegen

Smooth splitters

Definition 5 *A splitter S with m outputs is called smooth if its invariant* $\mathsf{Inv}(S)$ *can be specified by a collection of $m+1$ inequalities of the form*

$$d_0 x \leq f_0(\sigma)$$

$$d_i y_i \leq f_i(\sigma) \text{ for all } i,\ 1 \leq i \leq m\ ,$$

where for each i with $0 \leq i \leq m$, d_i is any of the four contention measures $\eth$, δ, Δ or ∇, and each f_i is a function mapping splitter states to integers.

Almost all splitters are smooth.

Examples (one shot)

- Aspnes *et al.* [AHS94] balancer,

$$\nabla y_1 \leq \left\lceil \frac{\nabla x}{2} \right\rceil \;\wedge\; \nabla y_2 \leq \left\lfloor \frac{\nabla x}{2} \right\rfloor ,$$

- Aspnes *et al.* [AHS94] counting network

For all i, $1 \leq i \leq m$: $\nabla y_i \leq \left\lceil \frac{\nabla x - i + 1}{m} \right\rceil ,$

- Aspnes *et al.* [AHS94] k-smoothing network

For all i, $1 \leq i \leq m$: $\nabla y_i \leq \min \left\{ \nabla y_j \mid j \neq i \right\} + k$.

- Moir *et al.* [MA95]

$$\nabla y_1 \leq 1 \;\wedge\; \nabla y_2 \leq \nabla x - 1 \;\wedge\; \nabla y_3 \leq \nabla x - 1$$

Examples (long-lived)

- Buhrman *et al.* [BGHM95]

  For all i, $1 \le i \le 3$: $\delta y_i \le \max(1, \delta x - 1)$.

- Afek *et al.* [AAF$^+$99]

  $$\delta y_1 \le 1 \;\wedge\; \nabla y_2 \le \nabla x - 1 \;\wedge\; \nabla y_3 \le \nabla x - 1\,,$$

- Moir *et al.* [MA95]

  $$\delta y_1 \le 1 \;\wedge\; \delta y_2 \le \delta x - 1 \;\wedge\; \delta y_3 \le \delta x - 1\,.$$

Impossibility results (1)

Theorem 6 *Let S be a splitter with $m > 1$ outputs. Suppose for some constant $c > 1$ we can select constants $c_1, \ldots, c_m$ such that for all states σ of S with $dx = c$ we have*

$$f_i^S(\sigma) \leq c_i$$

and

$$\sum_{i=1}^{m} c_i < c + \frac{m-1}{2} .$$

Then a read/write implementation of S does not exists

Impossibility results (2)

Proof (sketch):

- Consider one-shot case, and let c tokens enter.
- At each output y_i run renaming algorithm (e.g., [AF00]) to $2c_i - 1$ names.
- Then total number of assigned names is

$$\sum_{i=1}^{m} (2c_i - 1) = 2 \sum_{i=1}^{m} c_i - m$$

- Impossible if $< 2c - 1$ (Herlihy and Shavit [HS93]).

Impossibility results (3)

Theorem 7 *Define* $M = \{1, \ldots, m\}$*. Let* S *be a splitter with* $m > 1$ *outputs. Suppose there exists an index set* $I \subset M$ *such that for all states* σ *of* S *with* $dx > 0$ *we have*

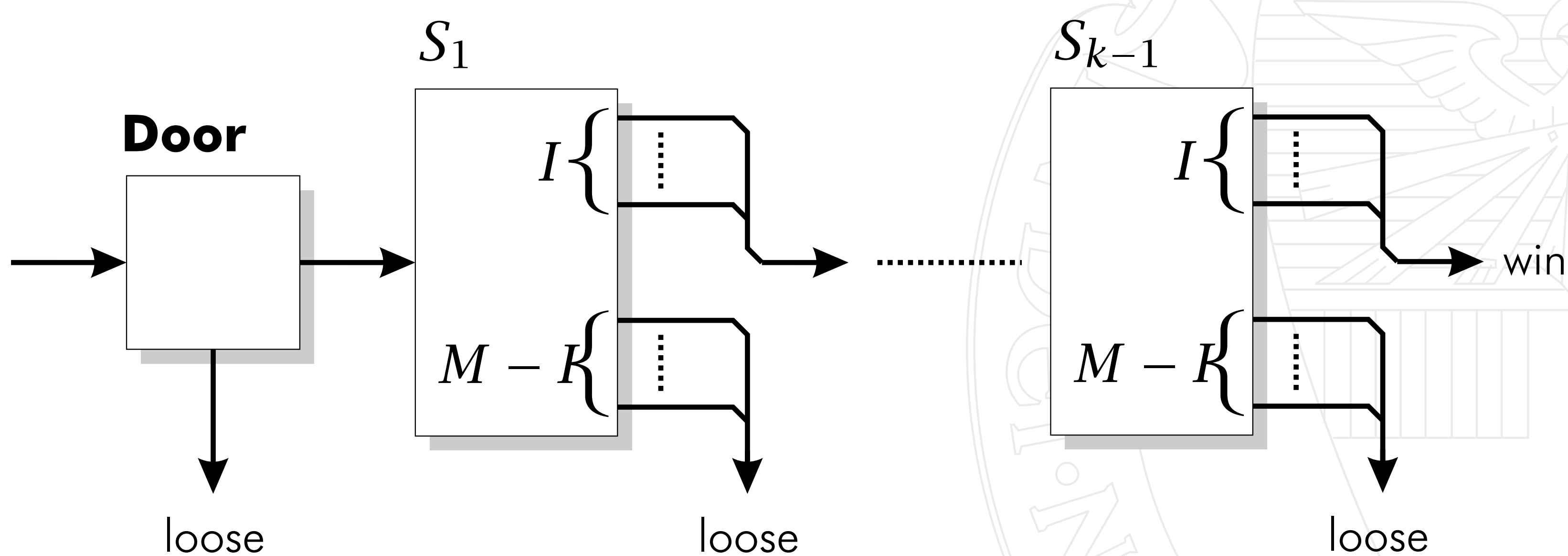
$$\sum_{i \in I} f_i(\sigma) < \max(2, dx) \quad \textit{and} \quad \sum_{i \in M-I} f_i(\sigma) < dx \;.$$

Then a read/write implementation of S *does not exist.*

Impossibility results (4)

Proof (sketch):

If both properties hold, we can build a test-and-set object. This contradicts [LAA87, Her91].



Possibility results (1)

Theorem 8 *Splitter S defined by*

$$\delta y_i \leq \frac{2}{3}\delta x, \quad \textit{for } 1 \leq i \leq 3.$$

has a read/write implementation.

Proof 9 *Use any optimal long-lived renaming algorithm (like [AF00]) to rename the δx incoming tokens to $2\delta x - 1$ names. Map a token with name i to output $y_{(i \bmod 3)+1}$. Then $\delta y_i \leq \frac{2}{3}\delta x$.*

University of Nijmegen

Possibility results (1)

Theorem 10 *Let S be a splitter satisfying the axioms, shared with n processors. This splitter can be implemented using a single n processor read-modify-write register.*

Conclusions

Results:

- Start of independent theory of splitters.
- Splitters can be defined using invariants in a straightforward pattern.
- RMW registers are strong enough to build splitters.
- But in read/write case, certain classes of splitters cannot be constructed.

Remaining issues:

- Building splitters using other splitters as building block.
- The place of splitters in Herlihy's hierarchy [Her91].

References

[AAF^{+}99] Afek, Y., Attiya, H., Fouren, A., Stupp, G., and Touitou, D. Long-lived renaming made adaptive. In *18th PODC* (Atlanta, GA, USA, 1999), ACM Press, pp. 91–103.

[AM94] Anderson, J. H., and Moir, M. Using k-exclusion to implement resilient, scalable shared objects. In *13th PODC* (Los Angeles, CA, USA, 1994), ACM Press, pp. 141–150.

[AHS94] Aspnes, J., Herlihy, M., and Shavit, N. Counting networks. *J. ACM* **41**, 5 (1994), 1020–1048.

University of Nijmegen

[AF00] Attiya, H., and Fouren, A. Polynomial and adaptive long-lived $(2k-1)$-renaming. In *14th DISC* (Toledo, Spain, 2000), M. Herlihy (Ed.), LNCS 1914, Springer, pp. 149–163.

[BGHM95] Buhrman, H., Garay, J. A., Hoepman, J.-H., and Moir, M. Long-lived renaming made fast. In *14th PODC* (Ottawa, Ont., Canada, 1995), ACM Press, pp. 194–203.

[Her91] Herlihy, M. P. Wait-free synchronization. *ACM Trans. Prog. Lang. & Syst.* **13**, 1 (1991), 124–149.

[HS93] Herlihy, M. P., and Shavit, N. The asynchronous computability theorem for t-resilient tasks. In *25th STOC* (San Diego, CA, USA, 1993), ACM Press, pp. 111–120.

[Lam87] Lamport, L. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.* **5**, 1 (1987), 1–11.

[LAA87] Loui, M. C., and Abu-Amara, H. H. Memory requirements for agreement among unreliable asynchronous processes. In *Advances in Computing Research*, F. P. Preparata (Ed.), vol. 4. JAI Press, Greenwich, CT, 1987, pp. 163–183.

[MA95] Moir, M., and Anderson, J. H. Wait-free algorithms for fast, long-lived renaming. *Science of Computer Programming* **25**, 1 (1995), 1–39.