Generating Random Variates II and Examples

Holger Füßler

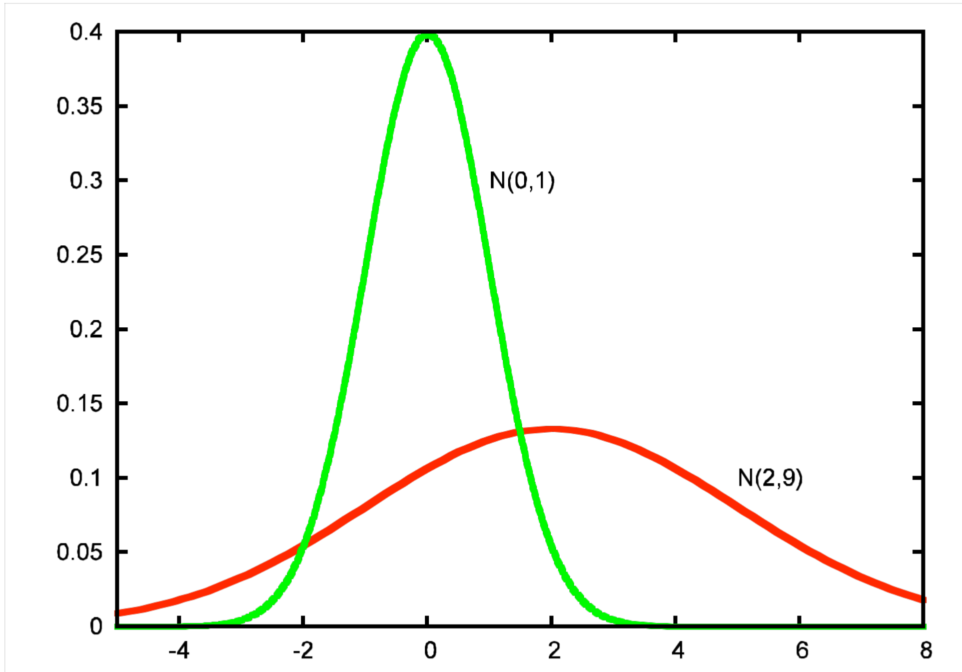
Side note: TexPoint

- TexPoint is a Powerpoint add-in that enables the easy use of Latex symbols and formulas in Powerpoint presentations.
- There are two main modes of operation: inline and display.
- In inline mode you can use Latex symbol-macro invocations such as "\alpha^2 \times \beta_0" on your Powerpoint slides.
- In the display mode you can write any Latex source and Latex is run to produce a bitmap that is then inserted on the slide. The bitmap remembers its Latex source so you can modify it later.
- http://raw.cs.berkeley.edu/texpoint/index.html

Course overview

1. Introduction
2. Building block: RNG
3. Building block: Generating random variates I and modeling examples
4. Building block: Generating random variates II and modeling examples
5. Algorithmics: Management of events
6. NS-2: Introduction
7. NS-2: Fixed networks
8. NS-2: Wireless networks
9. Output analysis: single system
10. Output analysis: comparing different configuration
11. Omnet++ / OPNET
12. Simulation lifecycle, summary

I Generation of normal variates



Density:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$

μ mean

σ standard deviation

Recommendation: Play with Gnuplot!

Given X ~ N(0,1) we can obtain X' ~ N(μ, σ^2) by setting X' = μ + σ X (see next slide).

Thus, we focus on N(0,1).

I Applications

- **Citing Law/Kelton:**

  "Errors of various types, e.g., in the impact point of a bomb;

  quantities that are the sum of large number of other quantities

  (by virtue of central limit theorem)."

- **Central limit theorem: let X_1, X_2, … be IID random variables with mean μ and variance $\sigma^2 < 1$:**

$$T_n := \frac{\sum_{j=1}^{n} X_j - n\mu}{\sigma\sqrt{n}} \approx N(0,1)$$

I Recap: general process of generating random variates

- **Formal algorithm - depends on desired distribution.**
- **But *all* algorithms have the same general form:**
  - **Generate one or more IID U(0, 1) random numbers**
  - **Transformation (depends on desired distribution)**
  - **Return *X* ~ desired distribution**

Optional: Read everything about Box-Muller-Transform and Polar Method

I Code example (NS-2)

tools/rng.cc:

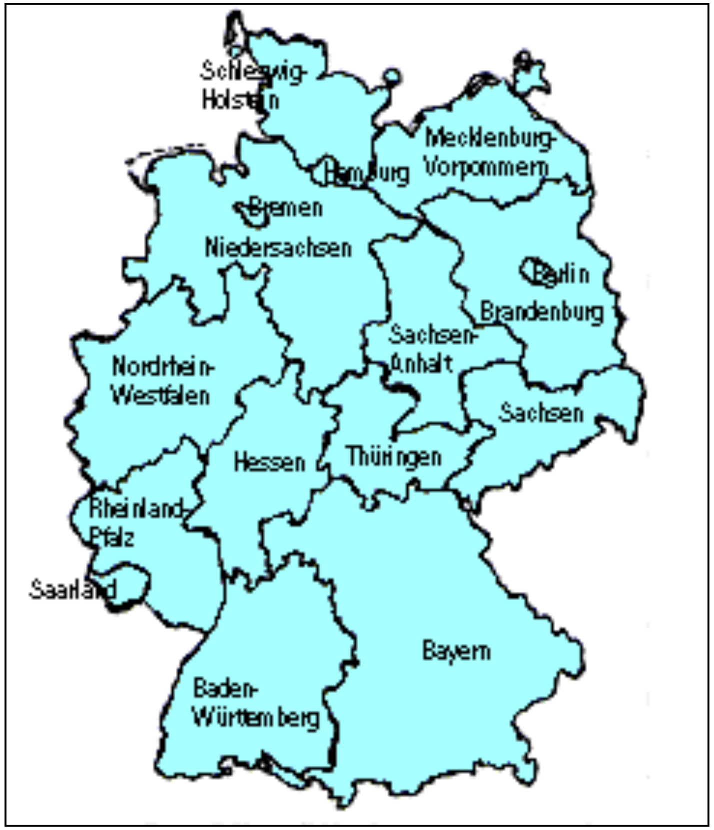
- Generate random numbers U_1 and U_2
- Set
  - $V_1 = 2\ U_1 - 1$
  - $V_2 = 2\ U_2 - 1$
  - $R^2 = V_1^2 + V_2^2$
- If $R^2 > 1$ return to step 1
- Return indep. normals

```
double
RNG::normal(double avg, double std)
{
        static int parity = 0;
        static double nextresult;
        double sam1, sam2, rad;

        if (std == 0) return avg;
        if (parity == 0) {
                sam1 = 2*uniform() - 1;
                sam2 = 2*uniform() - 1;
                while ((rad = sam1*sam1 + sam2*sam2) >= 1) {
                        sam1 = 2*uniform() - 1;
                        sam2 = 2*uniform() - 1;
                }
                rad = sqrt((-2*log(rad))/rad);
                nextresult = sam2 * rad;
                parity = 1;
                return (sam1 * rad * std + avg);
        }
        else {
                parity = 0;
                return (nextresult * std + avg);
        }
}
```

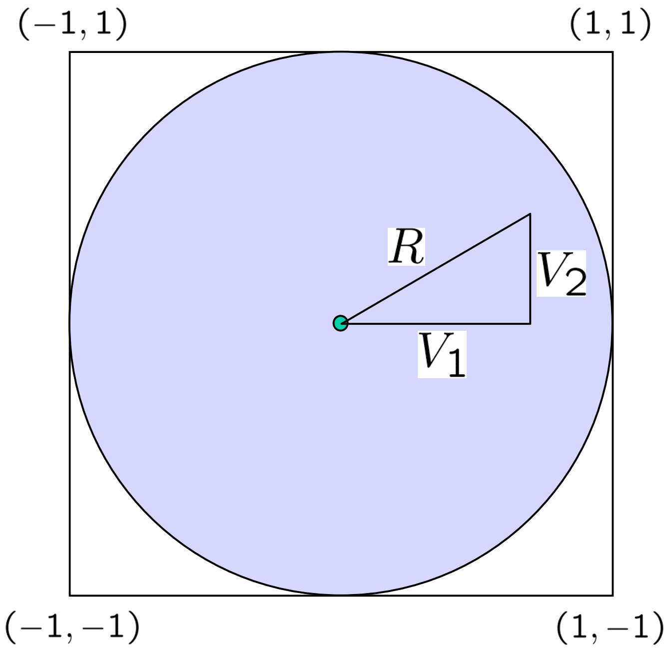
rad==0?

II Acceptance-rejection method: introduction



» **How to generate a uniform distribution for a (bounded) irregular shape?**

II Acceptance-rejection method: introduction



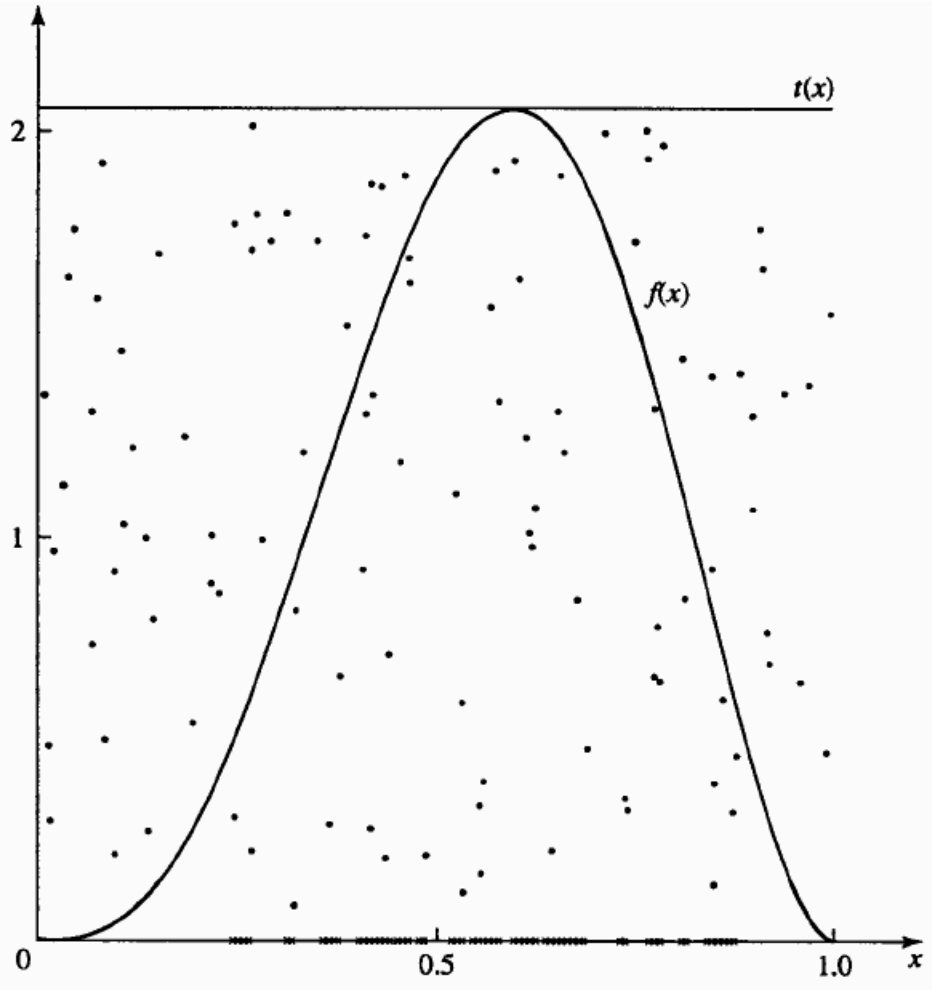
Start of algorithm:

- Generate random numbers U1 and U2
- Set
  - $V_1 = 2\,U_1 - 1$
  - $V_2 = 2\,U_2 - 1$
  - $R^2 = V_1^2 + V_2^2$
- If $R^2 > 1$ return to step 1

1. **Select a point in the 'bounding box' by sampling from a uniform distribution.**
2. **Check, if the point is in the shaded area:**
   1. If not, go to step 1.
   2. If yes, select it as output value.

Generates uniform distribution on shaded area.

II Acceptance-rejection method:



$f(x) = 60\, x^3\, (1 - x)^2$ for $0 \le x \le 1$

» **Do same thing as before, but take projection to x-axis as output value.**

II Acceptance-rejection method: general algorithm

Goal: generate random variate

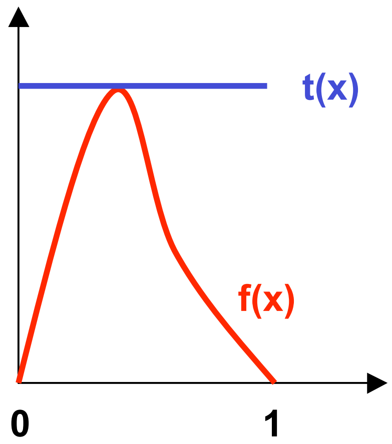
X with density function f(x).

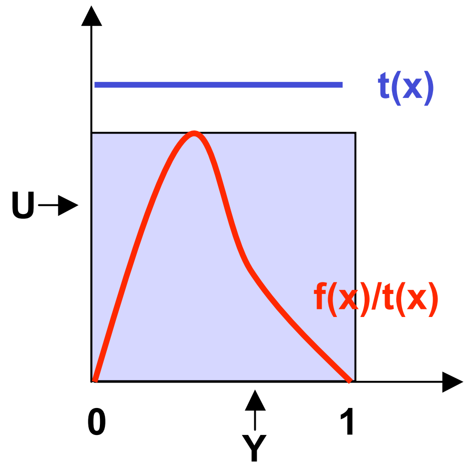
Specify function t(x) such that t(x) ≧ f(x)

for all x such that $c = \int_{-\infty}^{\infty} t(x)dx \geq \int_{-\infty}^{\infty} f(x)dx = 1$

Define r(x) = t(x)/c; it's a density.

Example:

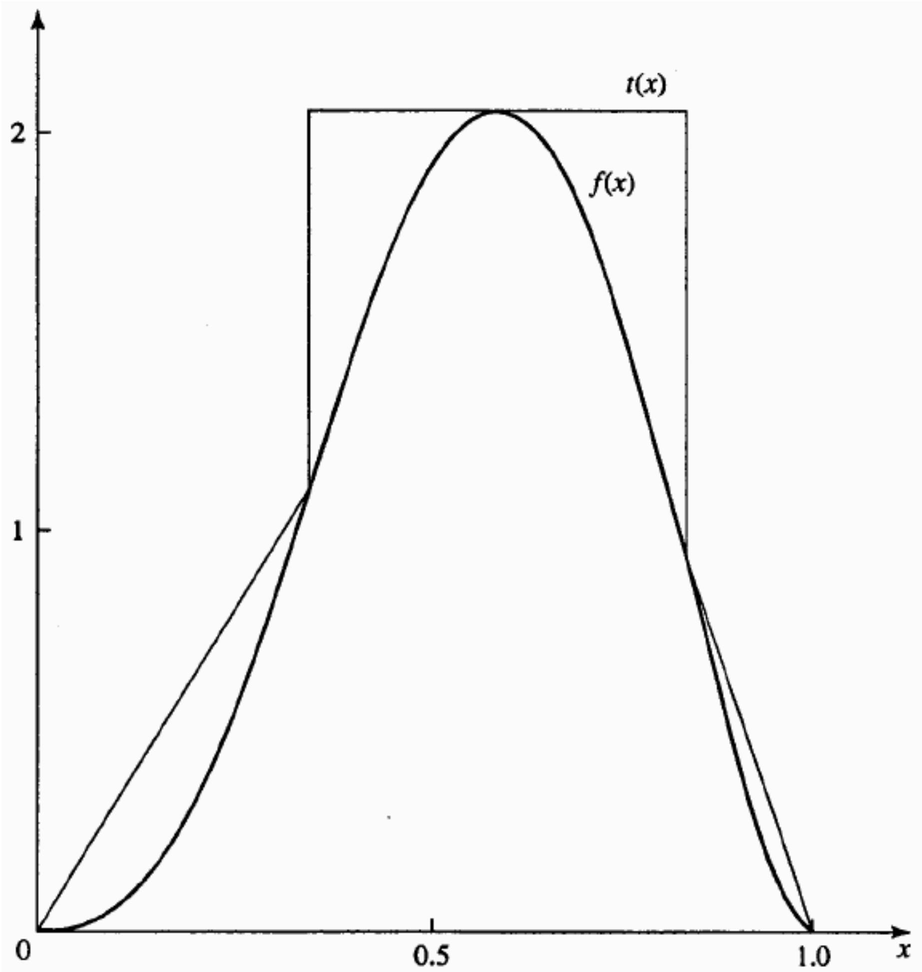




1. **Generate variate Y with density r(x).**
2. **Generate U~U(0,1) independent of Y.**
3. **If U ≤ f(Y)/t(Y), return X=Y and stop;**

   else go back to step 1.

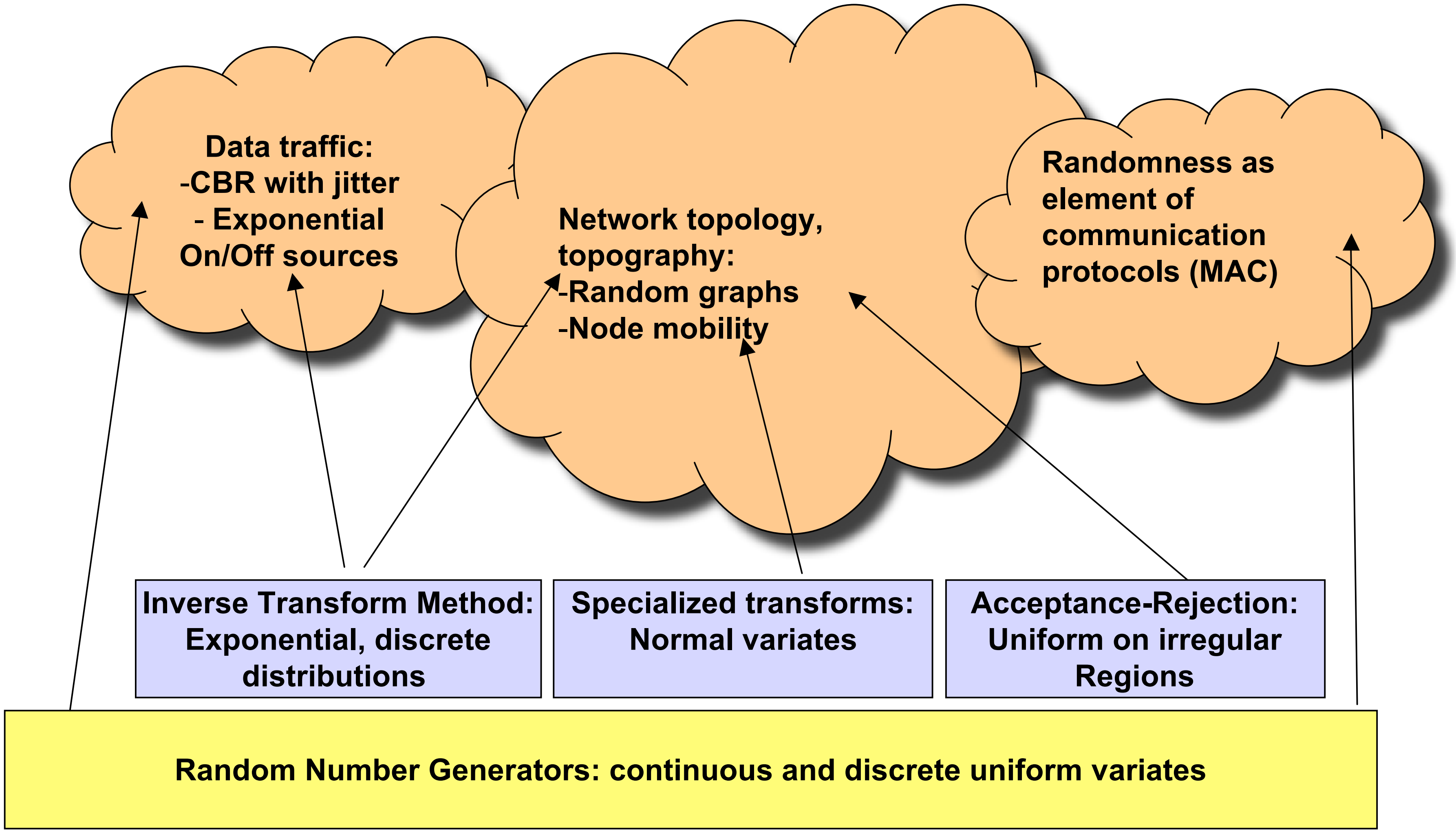
II Acceptance-rejection method: improvement



Wrap-up, summary

- U(0,1) → transform → desired distribution
- Normal variates frequently used to model ‘errors’ or variation of quantity
- Acceptance-rejection method
  - Less ‘direct’ than inverse transform method
  - Can be used when distribution function does not have closed form expression
  - Is used in polar method to generate uniform distribution on unit disc
- We know have all major stochastic building blocks for our simulations

So far … stochastic building blocks and models



References

- **Box-Muller and polar method:**
  - **Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., *Numerical Recipes in C*, 2nd edition, Cambridge University Press, 1992: chapter 7.**
  - **Knuth, D.E., *The Art of Computer Programming*, vol. 2, 3rd edition, Addison Wesley, 1998: chapter 3.**
  - **Ross, S. M.: *Simulation*, 2nd edition, Academic Press, 1997.**
- **Radio propagation models:**
  - **NS-2 Manual, Dec. 13, 2003, Chapter 18 'Radio Propagation Models'**
  - **Rappaport, T. S., *Wireless Communications – Principles and Practice*, 2nd. ed., Prentice Hall, 2002; Chapter 4.**
- **Acceptance-rejection method:**
  - **Averill M. Law, W. David Kelton: "Simulation Modeling and Analysis", McGraw-Hill, 3rd edition, 2000.**

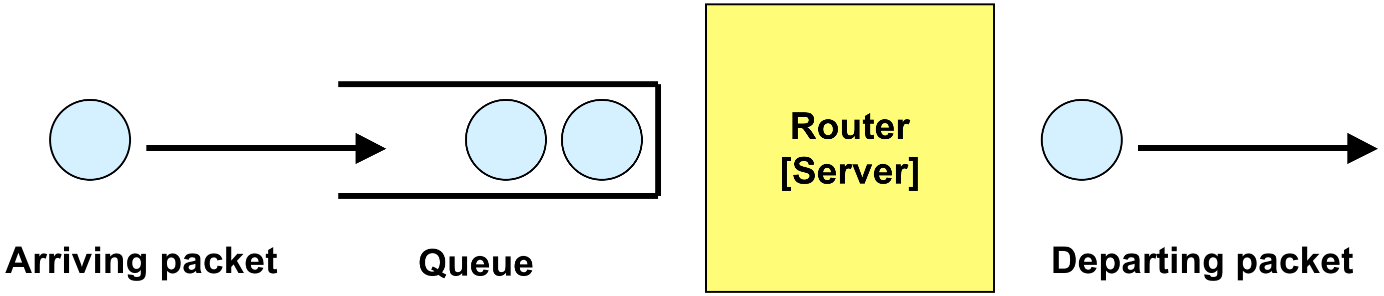
Event Scheduling

Holger Füßler

Course overview

1. Introduction	7. NS-2: Fixed networks
2. Building block: Random number generation	8. NS-2: Wireless networks
3. Building block: Generating random variates I and modeling examples	9. Output analysis: single system
4. Building block: Generating random variates II and modeling examples	10. Output analysis: comparing different configuration
5. Algorithmics: Event scheduling, management of events	11. Omnet++ / OPNET
6. NS-2: Introduction	12. Simulation lifecycle, summary

Recap: event scheduling w.r.t. M/M/1 queue



- Queuing systems as delay models
- Arrival process: 'M' for 'memoryless' (thus, exponentially distributed inter-arrival times)
- Service process: 'M' for 'memoryless' (thus, exponentially distributed service times)
- Number of queuing stations: 1

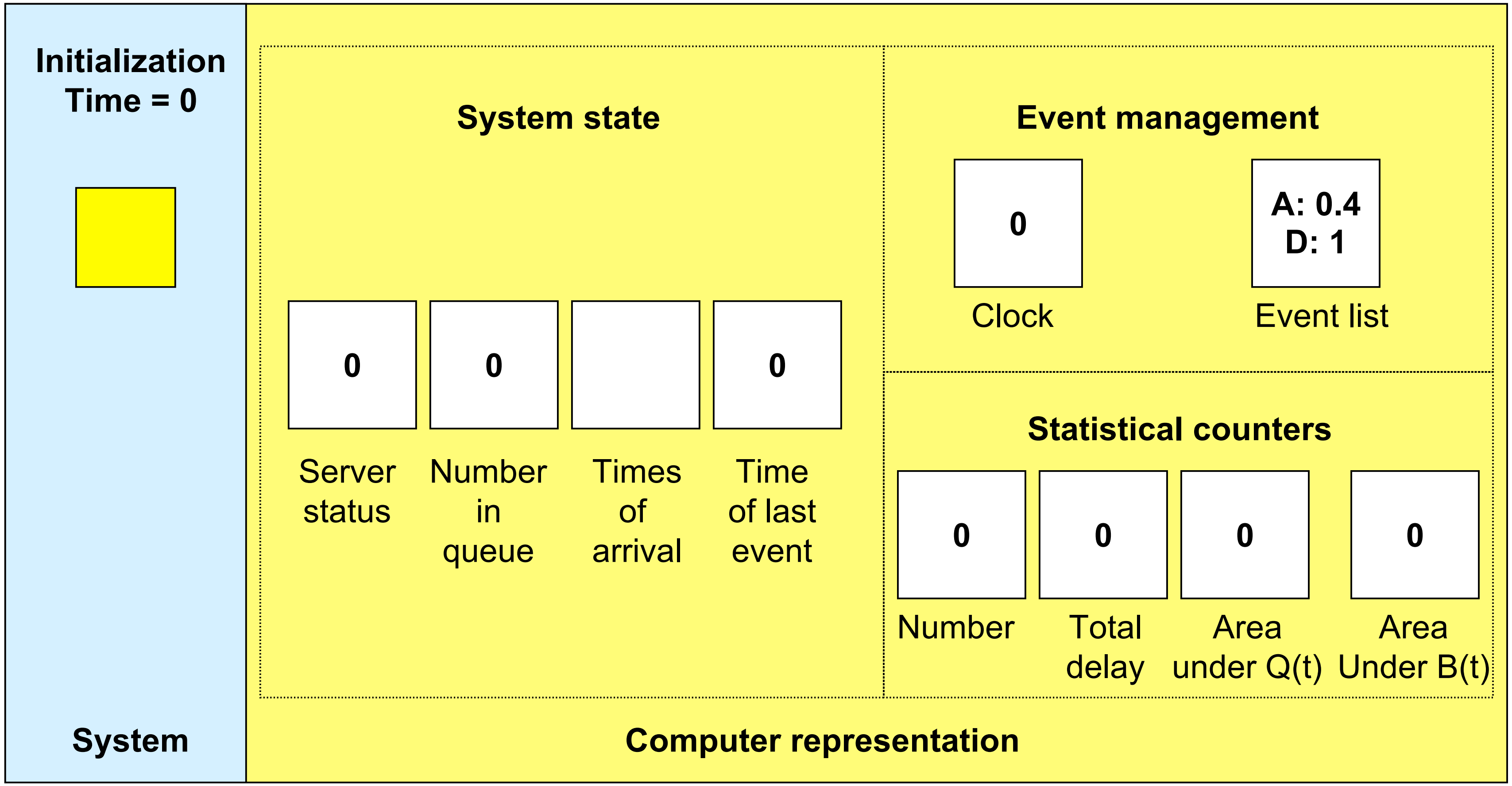
β=1.0 s for inter-arrival times

β=0.5 s for service times

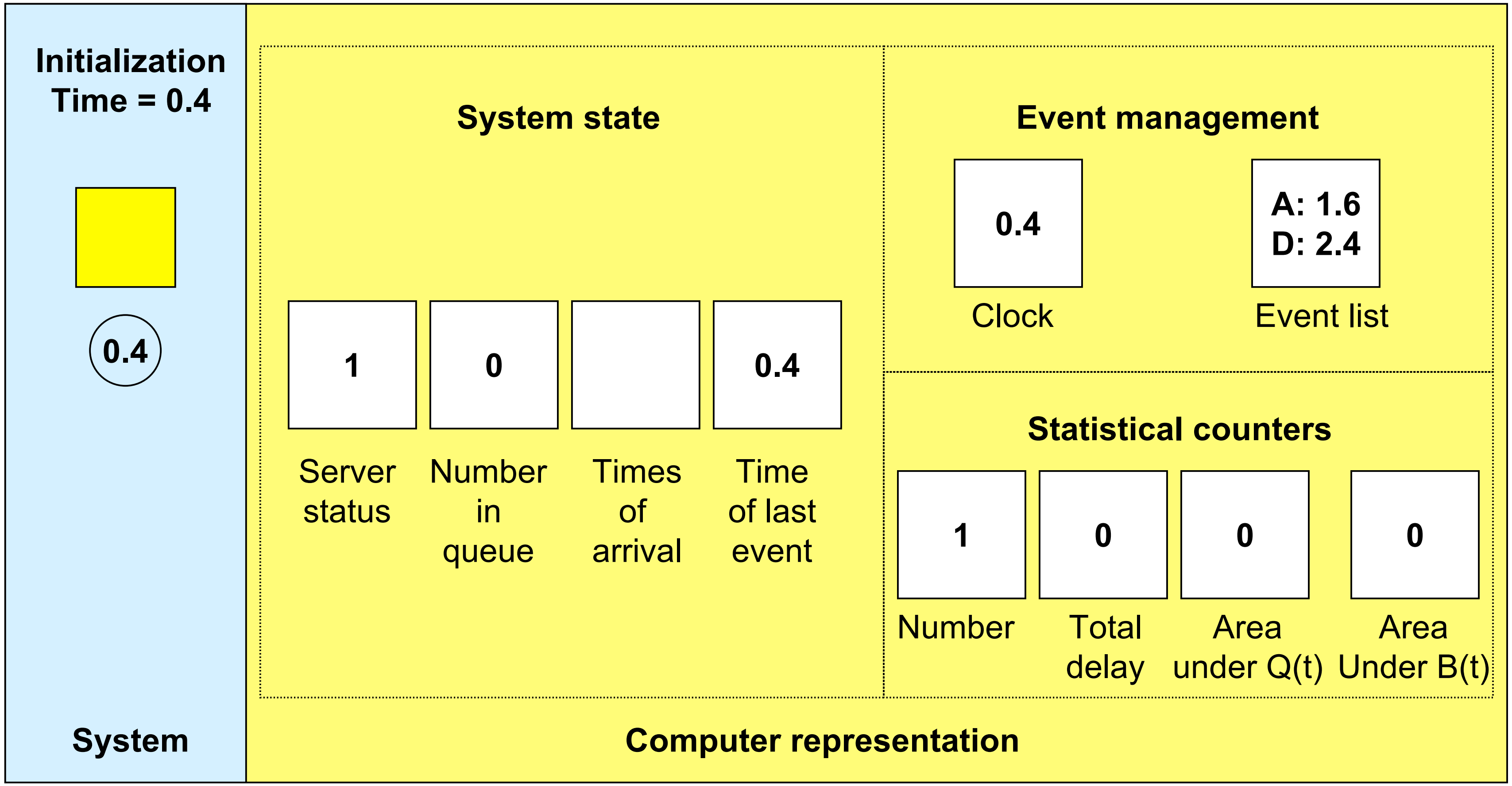
$$f(x) = \frac{1}{\beta} e^{-x/\beta}$$

- **Event: a state transition**
- **Event: depends on system logic and stochastic modeling**

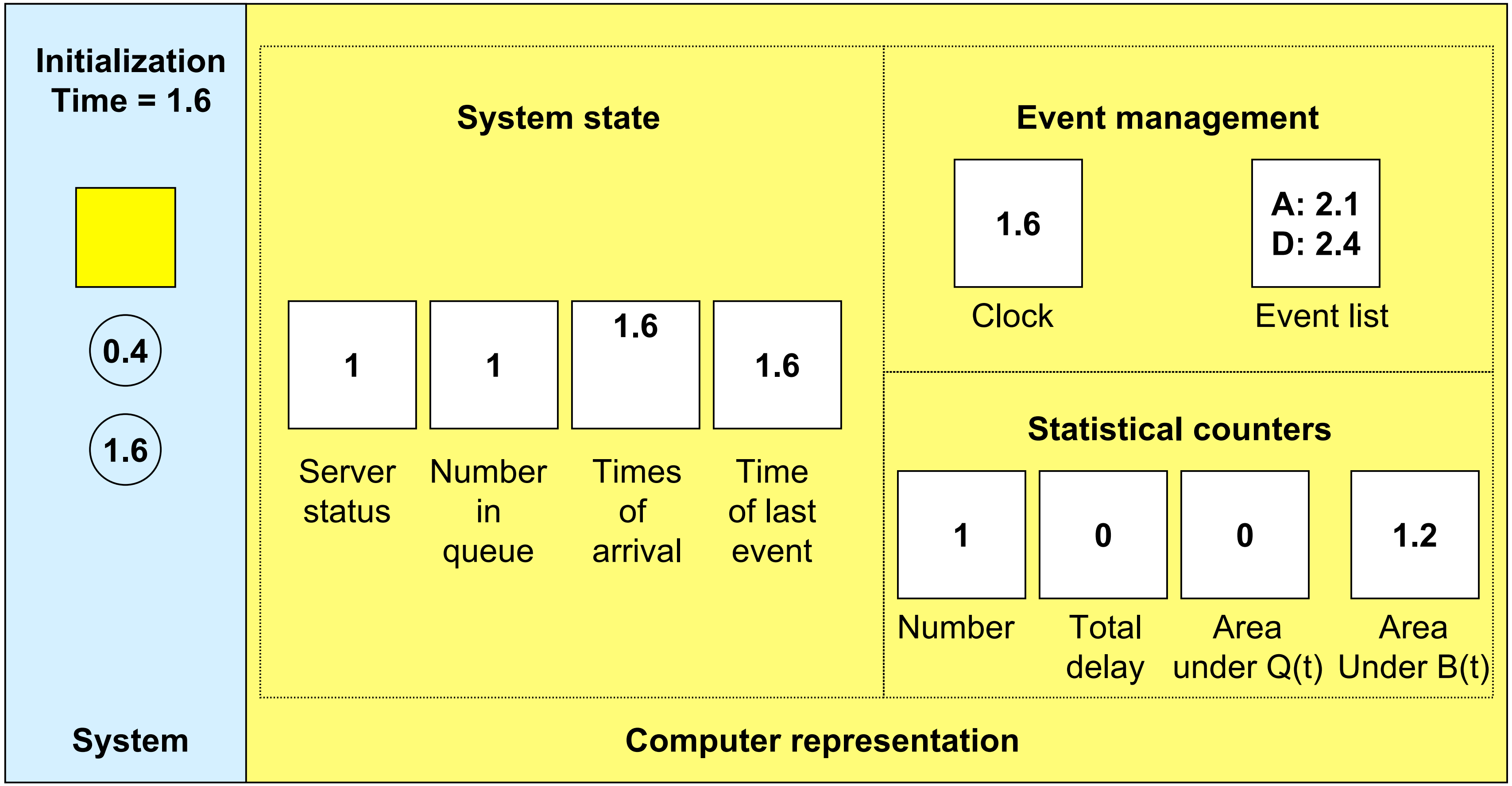
Recap: execute model



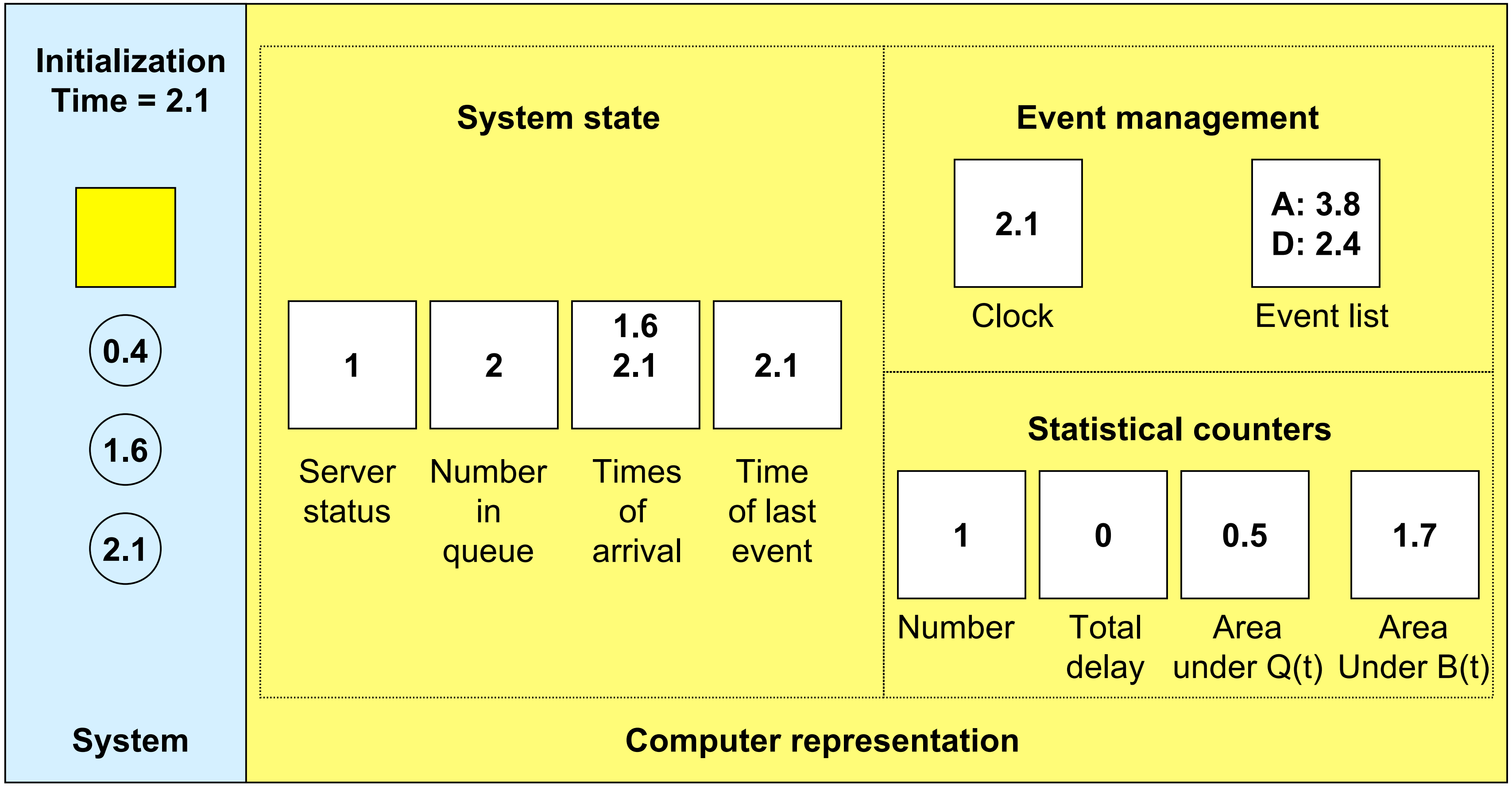
Recap: execute model



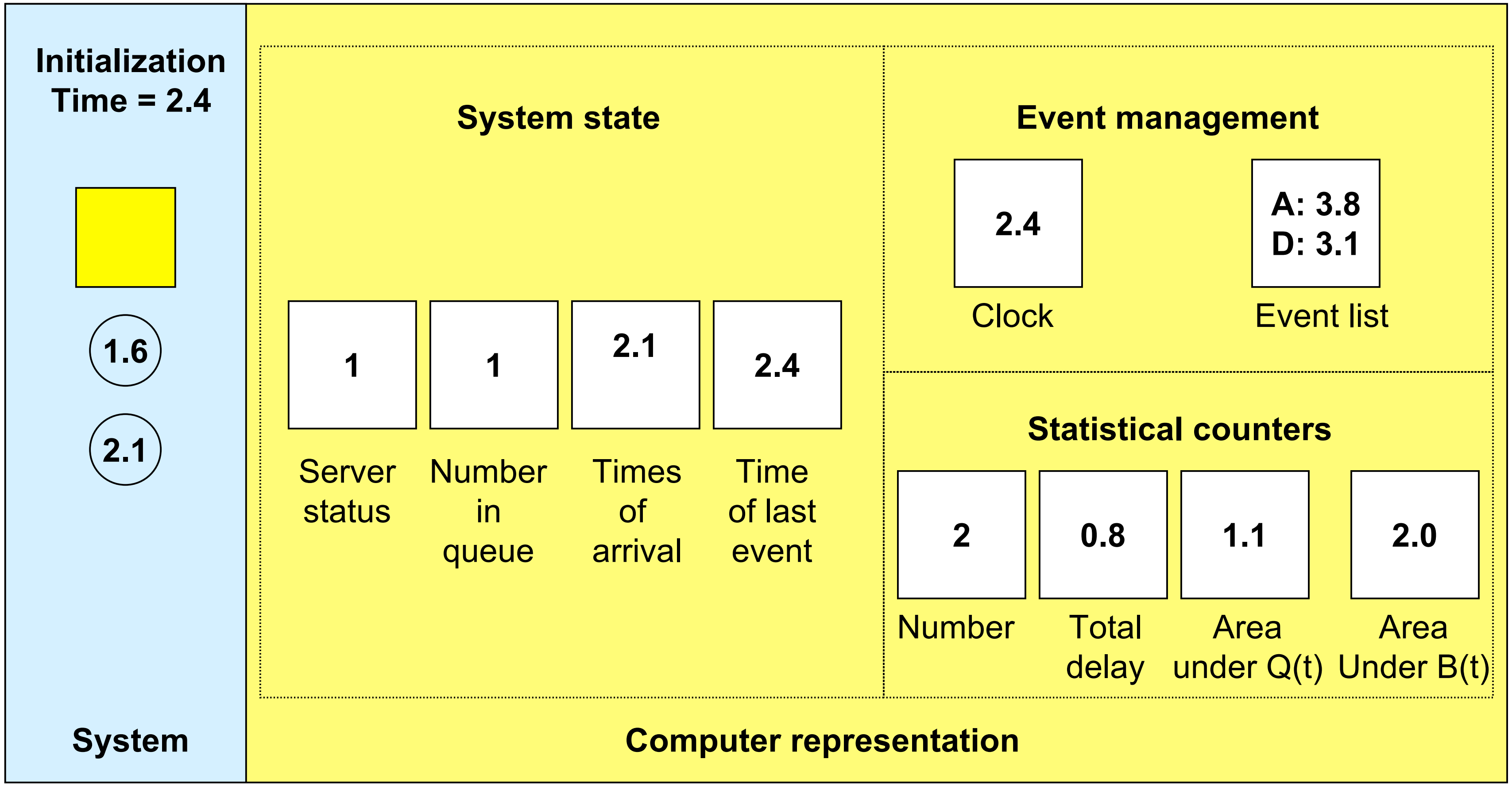
Recap: execute model



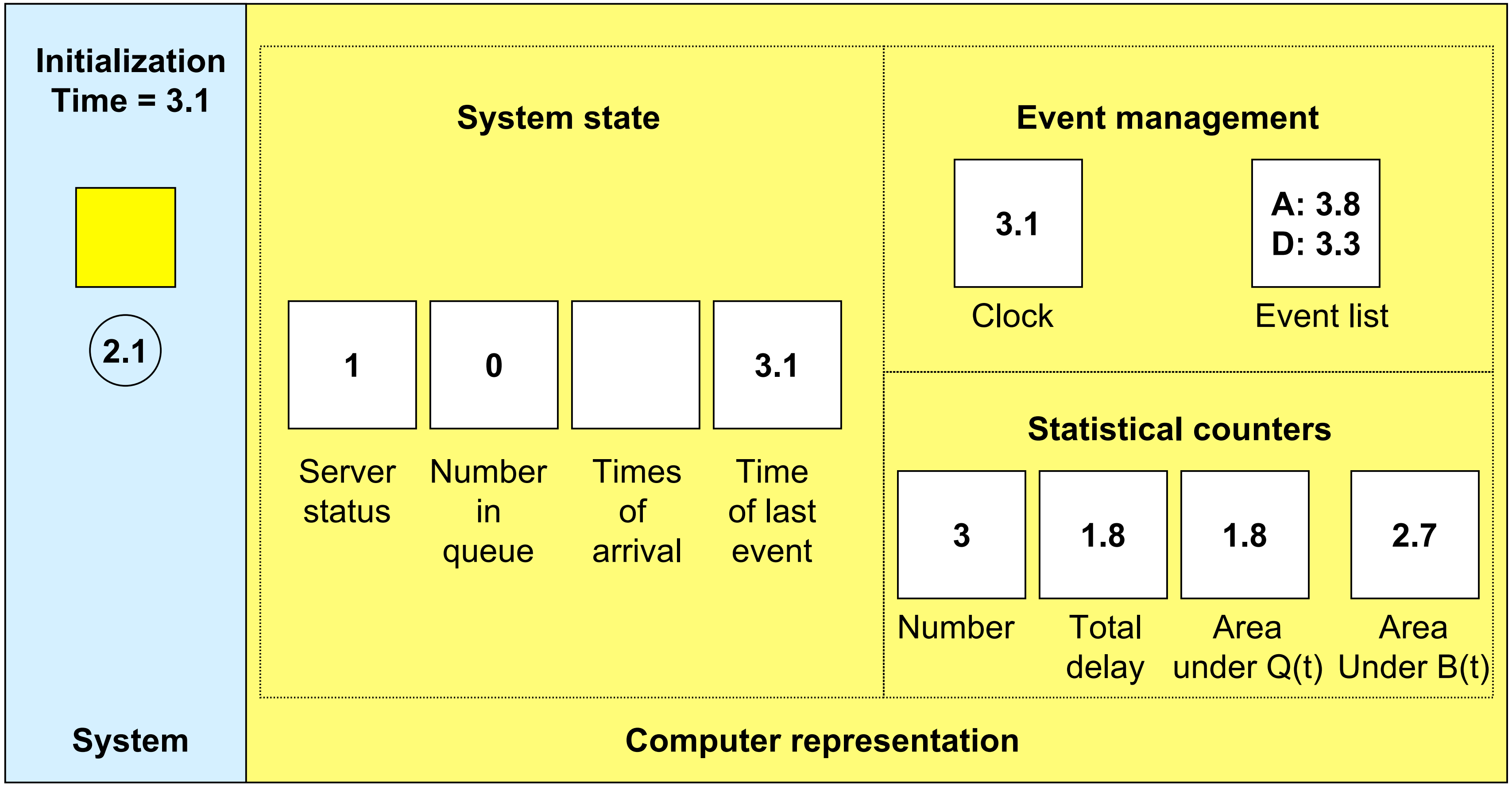
Recap: execute model



Recap: execute model



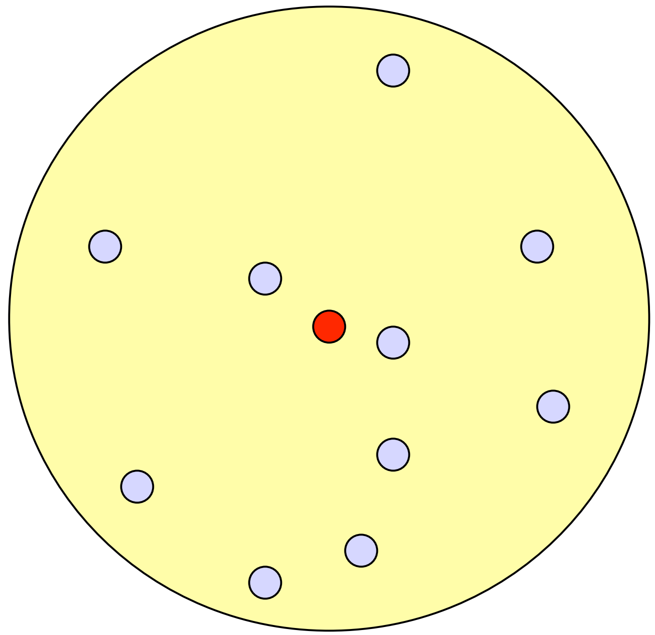
Recap: execute model



… see [LK2000] for continuation of this example

Scalability challenge

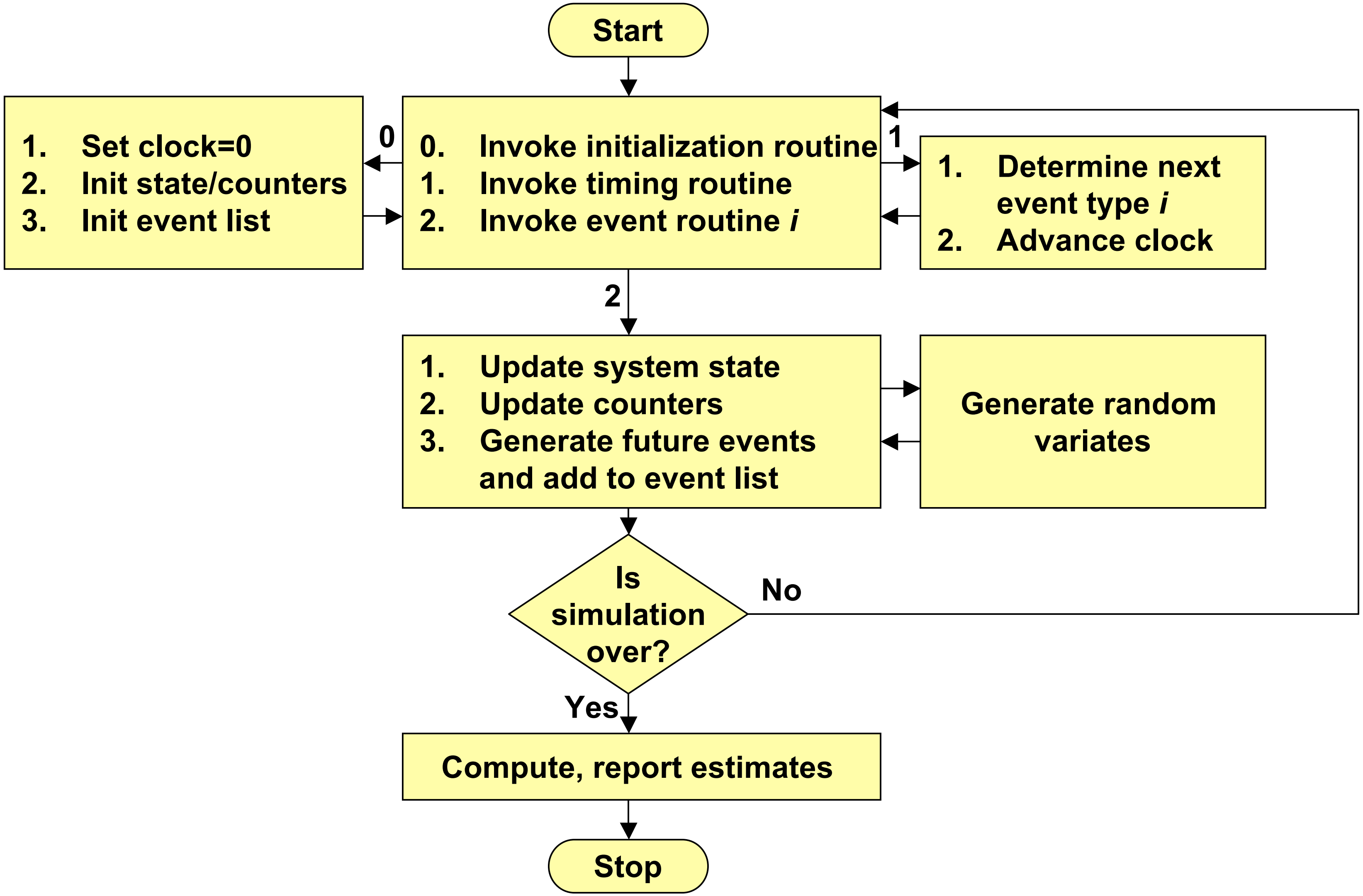
- Example: wireless communication
- For every packet send by a sender, all nodes (at least the ones within transmission range) have to schedule a ‘receive event’.
- Thus, we have at least $O(N^2)$ events where N denotes the number of nodes (sender/receiver).
- This lecture: focus on sequential processing of events



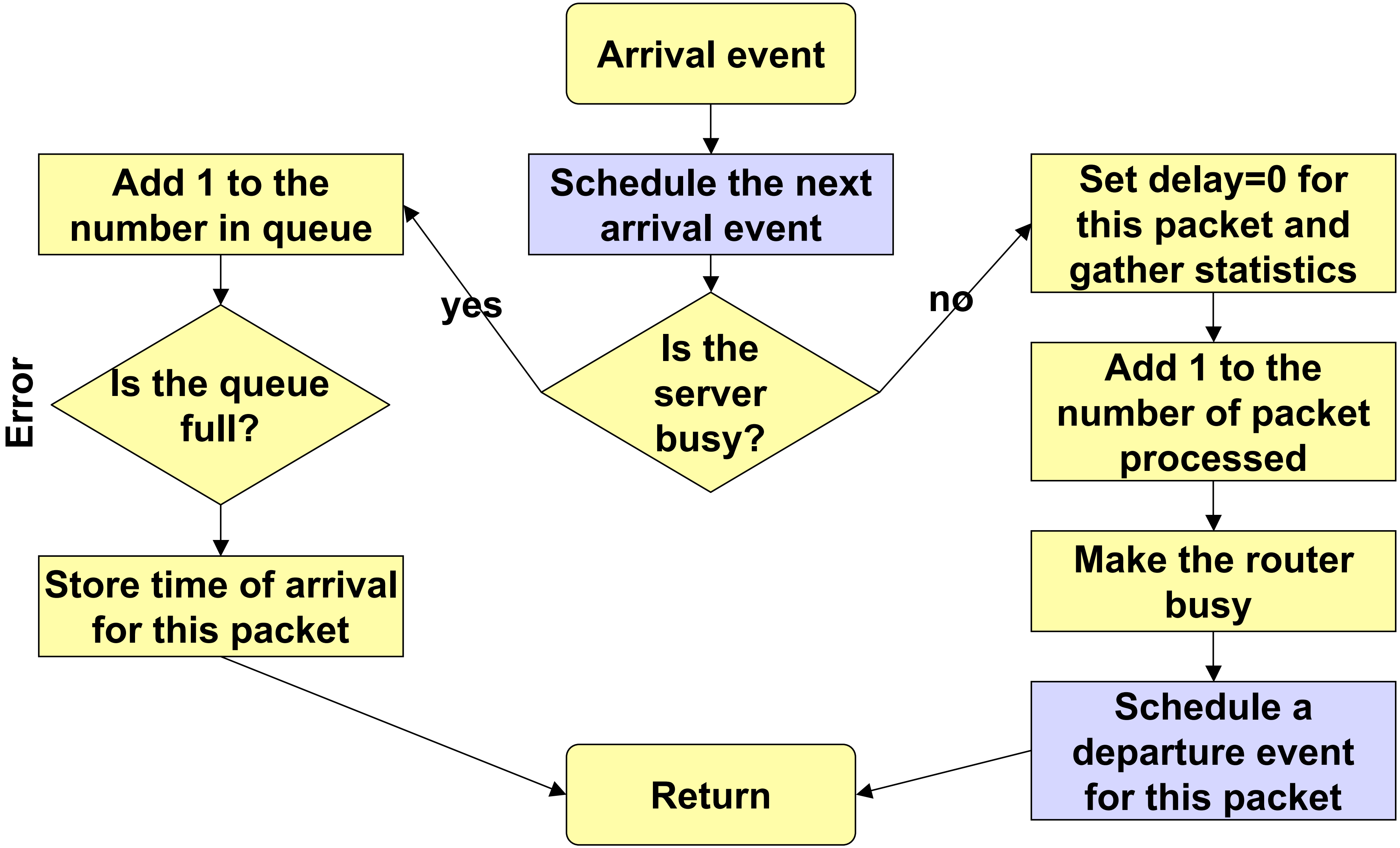
Structure

- Part I: Management of discrete events: problem statement
- Part II: Linear lists
- Part III: Heaps
- Part IV: Splay trees
- Part V: Calendar queue
- Part VI: Scheduling in NS-2

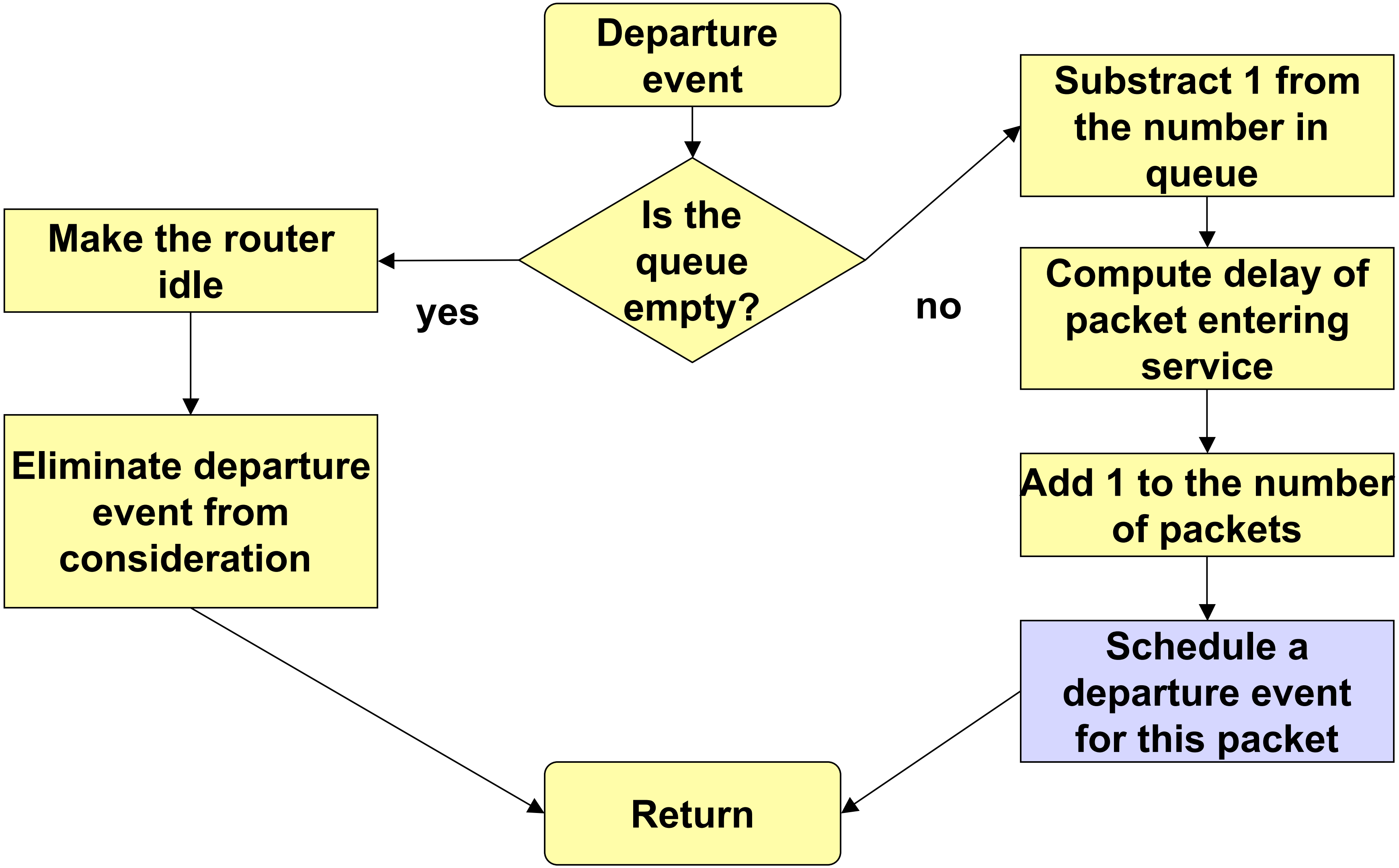
I Discrete event simulation: flow diagram



I Example M/M/1 queue: event type arrival



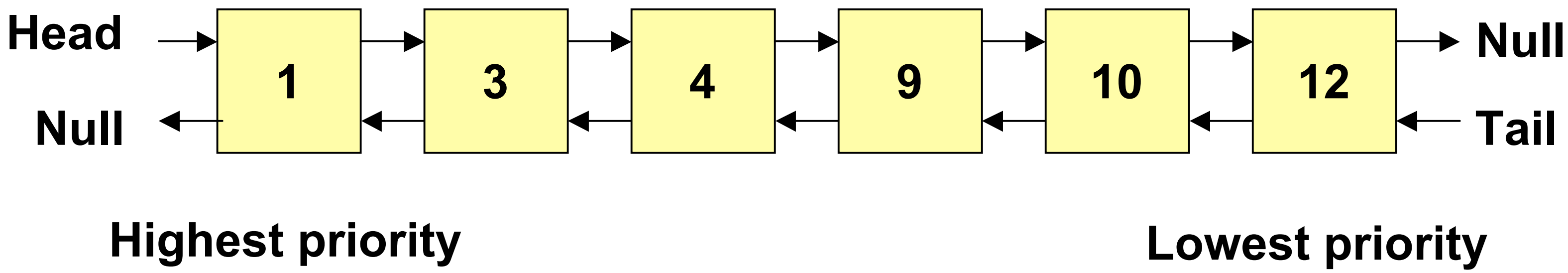
I Example M/M/1 queue: event type departure



I Event management: operations

- Enqueue event
- Dequeue ‘next’ event
- Usually: # enqueued events = # dequeued events
- But: distribution of event types and times can differ drastically
- Example:
  - Enqueue, enqueue, dequeue, enqueue, dequeue, enqueue, dequeue, …
  - Enqueue, enqueue, enqueue, … dequeue, dequeue, dequeue, …
- Required: efficient data structure for event management.
- Priority queues

II Sorted doubly-linked linear list



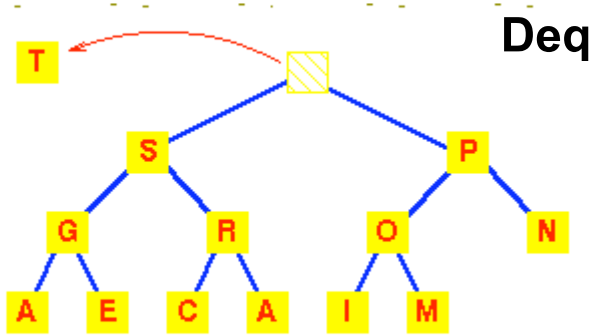
- Dequeue next event: take first element
  - Costs: O(1)
- Enqueue event according to priority
  - Costs: O(N) where N is the number of events in the list
- Knowledge on interval between events can be used to improve insertion process.

III Heaps

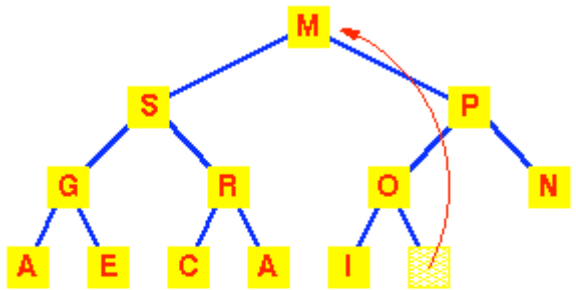
- Standard priority queue
- A binary tree has ‘heap property’ if
  - it is empty or
  - any node has a higher priority than its children
- Can be easily stored in an array
  - Root: A[1]
  - Children of A[i] are given by A[2i] (left) and A[2i+1] (right)
- Dequeue:
  - Remove root
  - Put element of current right bound of array to root
  - Restore heap property
  - Costs: O(log(N))
- Enqueue:
  - Put new element on current right bound of array
  - Restore heap property
  - Costs: O(log(N))

III Heaps: en-/dequeue

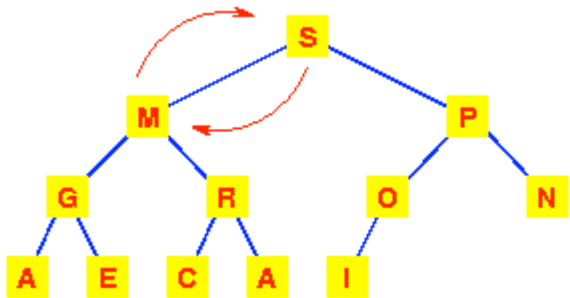
Dequeue



1

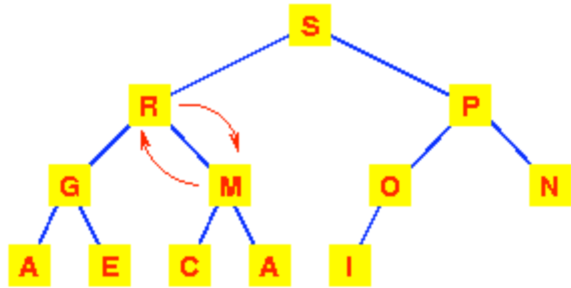


2

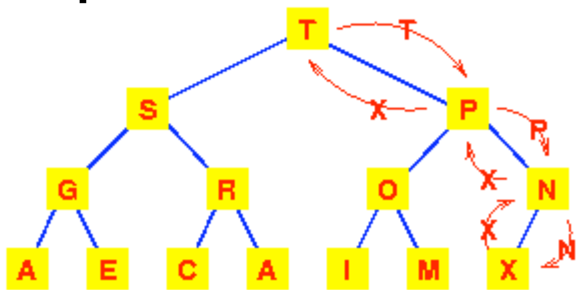


3

4



Enqueue

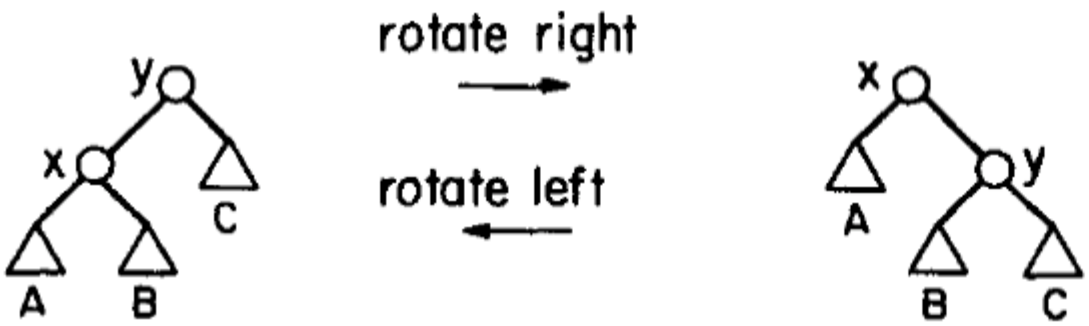


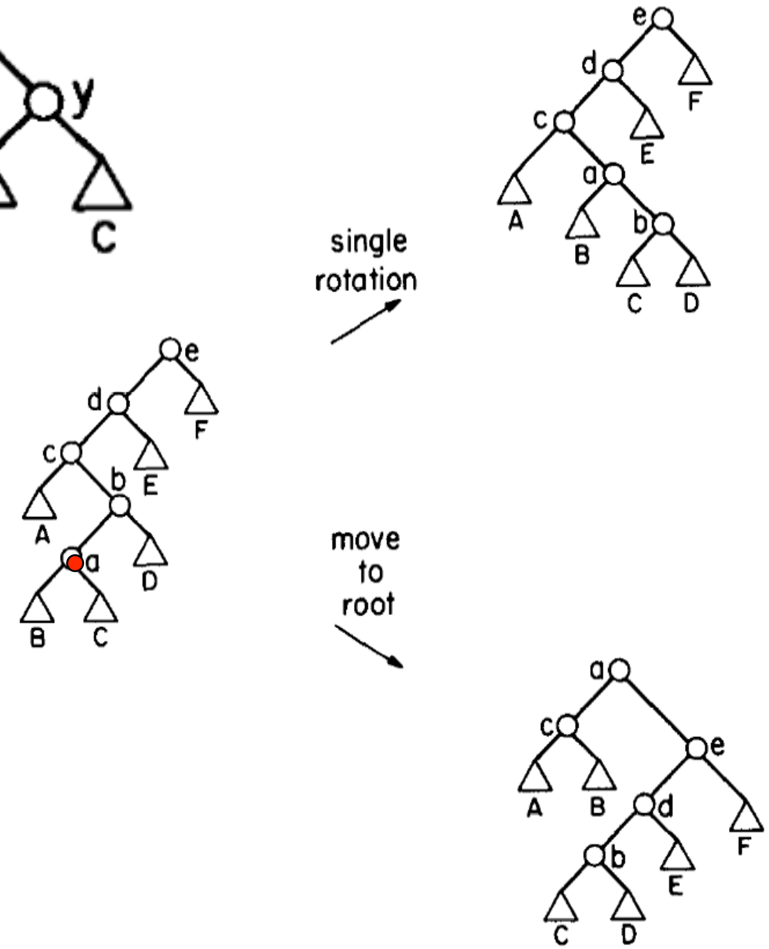
[Source:http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/heaps.html]

IV Splay trees

- Splay trees are binary search trees
- Binary search tree: for each node i
  - All nodes in left subtree of node i have smaller priority
  - All nodes in right subtree of node i have higher priority
- Costs for enqueue, dequeue in a sufficiently balanced binary search tree: O(log(N)) where N denotes the number of nodes
- Splay trees: use heuristics to reorg the tree during an en-/dequeue operation. The reorg pays off in subsequent operations.
- Implementation available in NS-2

IV Splaying operations (examples)





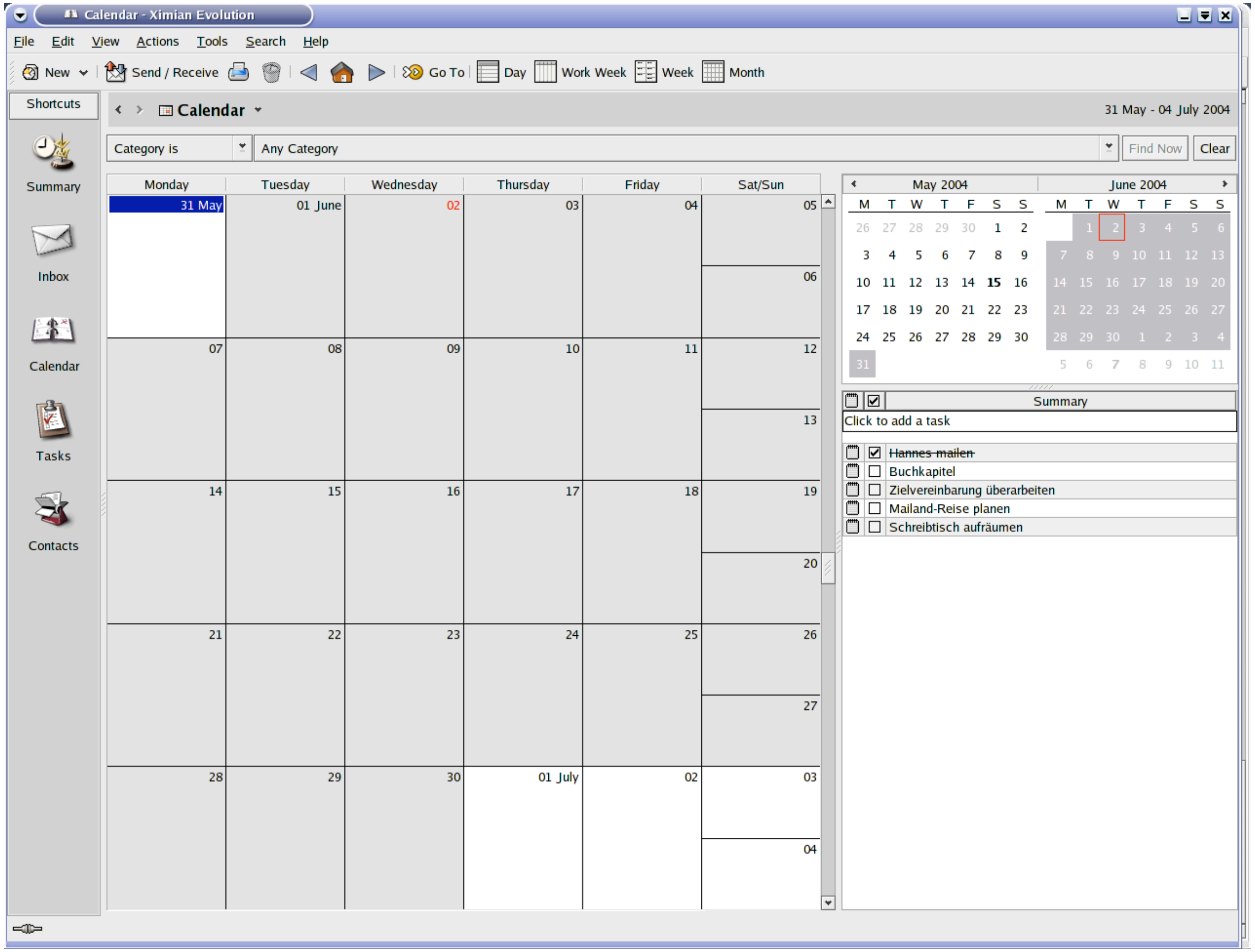
IV Results (Jones, 1986)

TABLE II. Summary of Conclusions

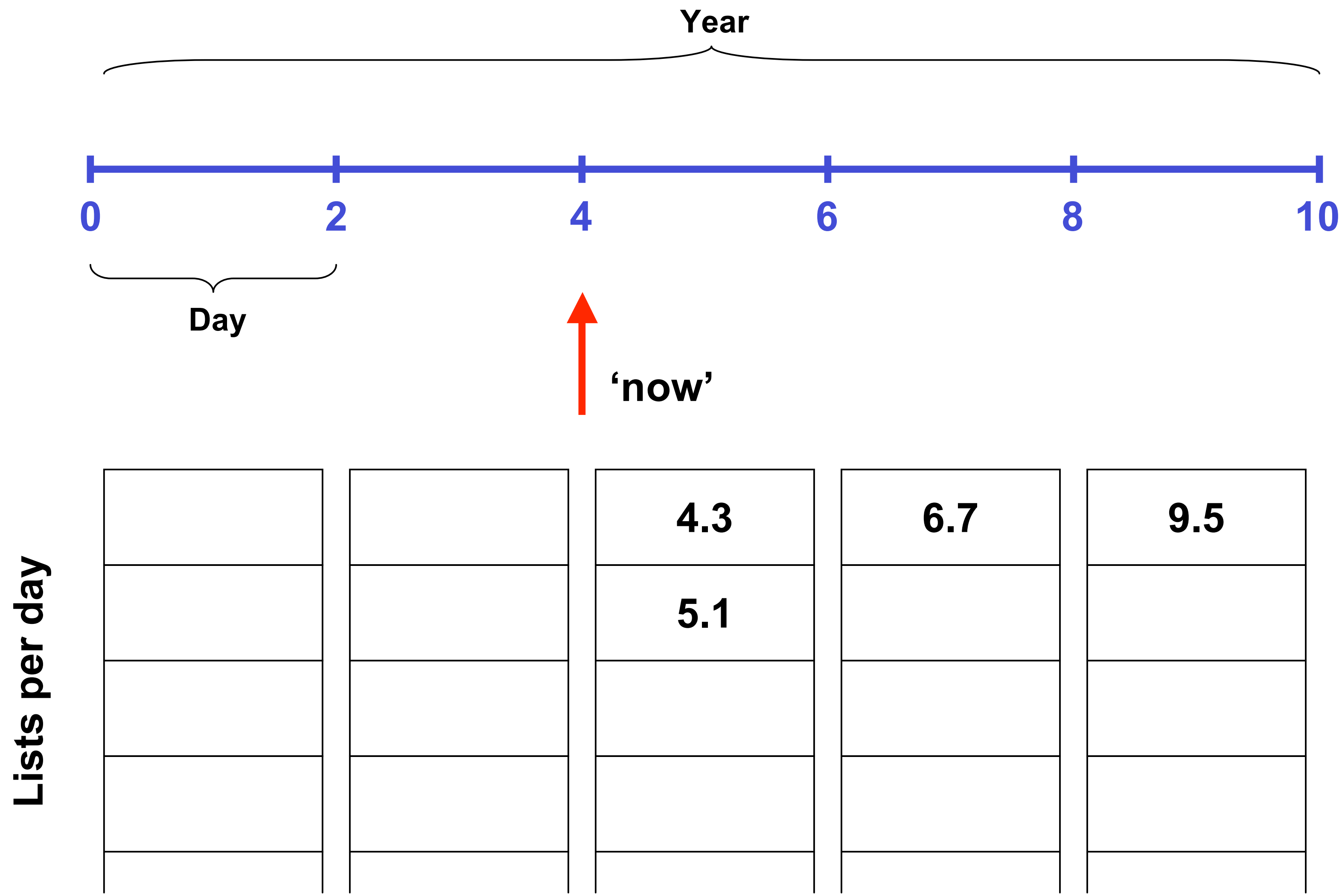
Priority-queue implementation	Code size[a]	Performance Average	Worst	Relative speed[b]	Comments
Linked list	47	$O(n)$	$O(n)$	11	Best for $n < 10$
Implicit heap	72	$O(\log n)$	$O(\log n)$	8	
Leftist tree	79	$O(\log n)$	$O(\log n)$	9–10	
Two list	104	$O(n^{0.5})$	$O(n)$	9–10	Good for $n < 200$
Henriksen's	68	$O(n^{0.5})$	$O(n^{0.5})^c$	1–7	Stable
Binomial queue	188	$O(\log n)$	$O(\log n)$	1–7	
Pagoda	110	$O(\log n)$	$O(n)$	4–8	Delete in $O(\log n)$
Skew heap, top down	56	$O(\log n)$	$O(\log n)^c$	5–7	
Skew heap, bottom up	103	$O(\log n)$	$O(\log n)^c$	4–6	Delete in $O(\log n)$
Splay tree	119	$O(\log n)$	$O(\log n)^c$	1–3	Stable
Pairing heap	84	$O(\log n)$	$O(\log n)^c$	3–6	Promote in $O(1)$

[a] The total lines of Pascal code for initqueue, emptyqueue, enqueue, and dequeue.
[b] 1 is fastest; 11 is slowest: The rankings are based on Figures 12–14.
[c] An amortized bound; single operations may take $O(n)$ time!

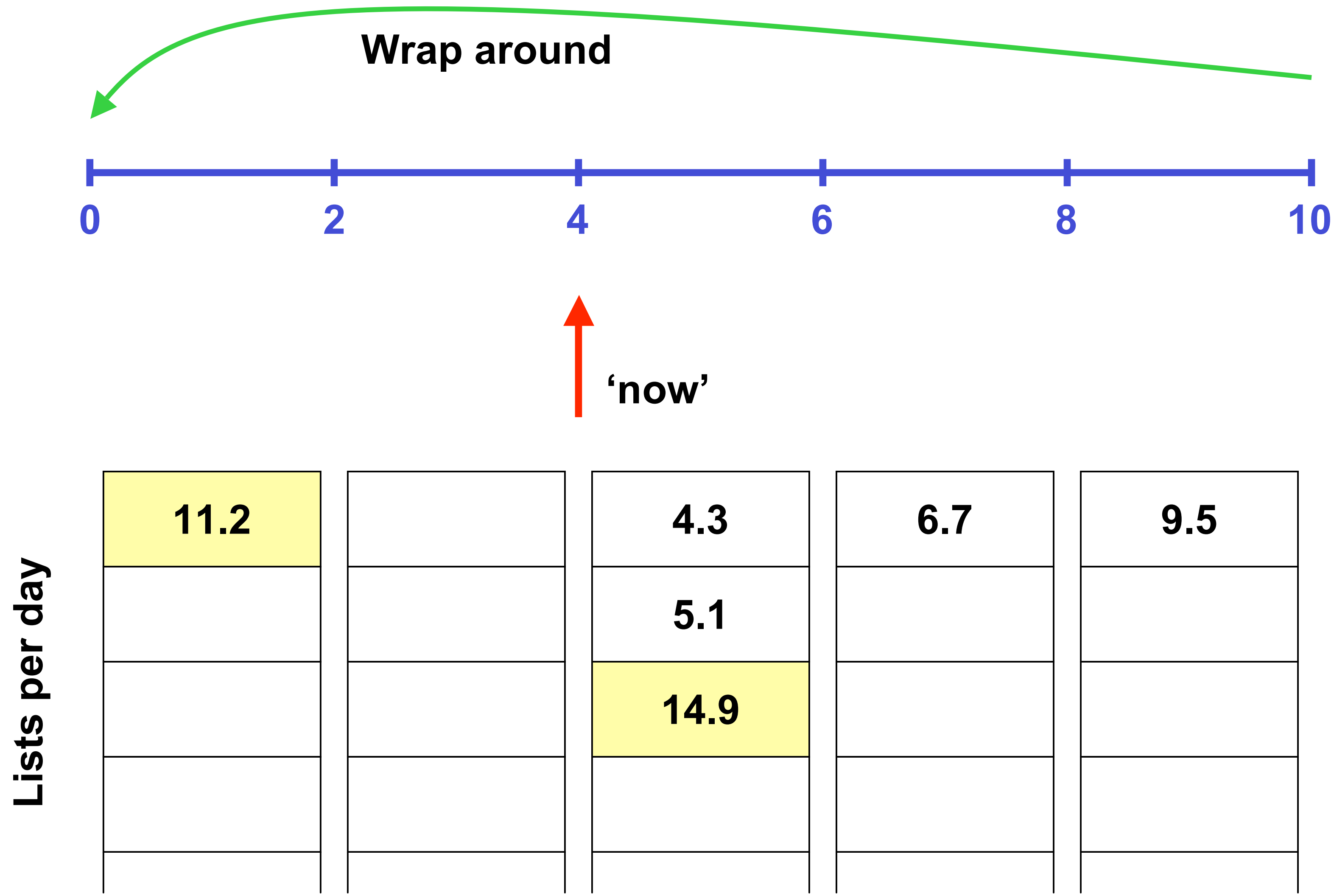
V Calendar queue: basic idea



V Calendar queue: static case I



V Calendar queue: static case II



V Calendar queue: static case III

Operations:

» **Get next event (dequeue):**
- **a) go to next event of current day or go to next day**
- **b) if a) fails for a whole year, use direct search**

Example:
- **500 events between [0,0.1]**
- **500 events between [5.6,5.7]**
- **Length of day: 0.0002**
- **Length of year: 0.1024 (512 days)**
- **Strategy a) needs to cycle through the calendar for 54 year to find the first event of [5.6,5.7]**

» **Insert *key* (enqueue):**
- **Find right day/bucket: *key* % ndays**
- **Insert *key* into ordered list**

V Problems with static case

- Many keys per day: enqueue operation gets expensive
  - Go through list of length O(N) where N denotes the number of events
- Only a few keys within many days: dequeue operation gets expensive
  - Go through empty days of length O(B) where B denotes the number of buckets
- Analysis of problem complexity is not trivial:
  - See ‘Optimizing static calendar queues’, K. B. Erickson, R. E. Ladner, A. Lamarca, ACM Tomacs, vol. 10, no. 3, July 2000, pp. 179-214.

V Calendar queue: dynamic case

Idea: Adjust length of year and days according to current key set.

» **Number of days (buckets):**

- **Whenever the number of keys exceeds twice the number of days, copy calendar to a larger calendar (typically: double size).**
- **Whenever the number of keys is less than half the number of days, copy to a smaller calendar (typically: half size).**

» **Length of day:**

- **Dequeue samples from calendar (typically 25)**
- **Calculate average separation of dequeued events**
- **Set new length of day to average separation (usually multiplied with some factor)**
- **Enqueue samples**

V Evaluation of calendar queues I

- Assume exponential distribution:
  - Mean 1
  - Next priority:
    last priority – ln(rng())
- Hold operation: dequeue followed by enqueue

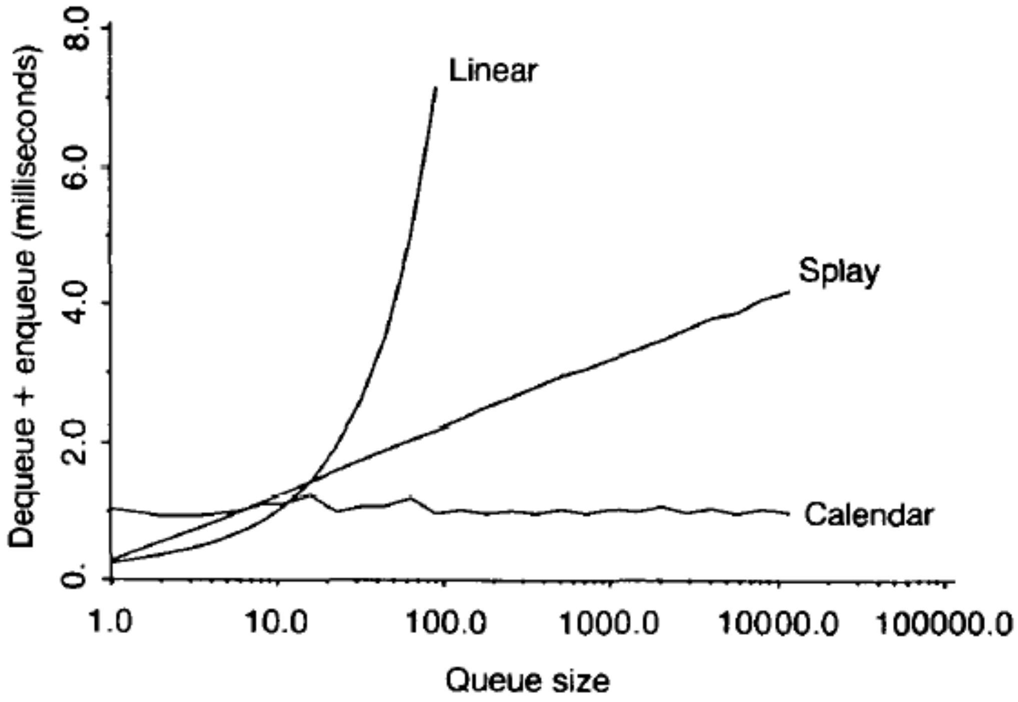


FIGURE 3. Comparison of Hold Times on Texas Instruments PC

[Source: Original paper by Brown, 1988]

VI Scheduling in NS-2

- **Choices are: list, heap, splay tree, calendar queue, real-time**
  - **Real-time: tries to synchronize with real-time clock; experimental**
- **Calendar queue is default**
- **How to deal with events at same time?**
  - If more than one event are scheduled to execute at the same time, their execution is performed on the first scheduled – first dispatched manner.
  - Simultaneous events are not reordered anymore by schedulers (as it was in earlier versions) and all schedulers should yield the same order of dispatching given the same input.
- **Accuracy: see next slide**

VI Precision of the scheduler clock in NS-2

» Precision of the scheduler clock: smallest time-scale of the simulator that can be correctly represented.

» The clock variable for ns is represented by a double. As per the IEEE std for floating numbers, a double, consisting of 64 bits must allocate the following bits between its sign, exponent and mantissa fields.

»

sign	exponent n	mantissa X
1 bit	11 bits	52 bits

» Any floating number can be represented in the form X ¢ 2^n where X is the mantissa and n is the exponent.

» Thus the precision of time in ns can be defined as $1/(2^{52})$.

» As simulation runs for longer times the number of remaining bits to represent the time educes thus reducing the accuracy. Given 52 bits we can safely say time up to around 2^{40} can be represented with considerable accuracy.

VI Precision of NS-2

- 2^{40} ¼ 10^{12}
- Thus, for a simulated time of 1000 seconds we still have nanosecond accuracy (10^{-9}).

Wrap-up

- Event management requires efficient priority queues: schedule an event, dequeue next event.
- Linear lists, heaps, splay trees, calendar queues
- Efficiency depends on ‘priority increment distribution’
- Calendar queue in general ‘safe guess’
- But: one should check its own priority increment distribution to see whether improvements are feasible
- Precision: with a 64-bit double, there is still a nanosecond accuracy for a simulated time of 1000 seconds.
- Open: is there a paper available with a performance evaluation of priority queues on current Pentium machines?

References

» **William M. McCormack, Robert G. Sagent, *Analysis of future event set algorithms for discrete event simulation*, Communications of the ACM, vol. 24, no. 12, December 1981, pp. 801-812**

» **Douglas W. Jones, *An empirical comparison of priority-queue and event-set implementations*, Communications of the ACM, vol. 29, no. 4, April 1986, pp. 300-311**

» **Randy Brown, *Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem*, Communications of the ACM, vol. 31, no. 10, October 1988, pp. 1220-1227**

» **Daniel D. Sleator, Robert E. Tarjan, *Self-adjusting binary search trees*, Journal of the ACM, vol. 32, no. 3, July 1985m pp. 652-686**

» **NS manual, Dec. 13, 2003.**