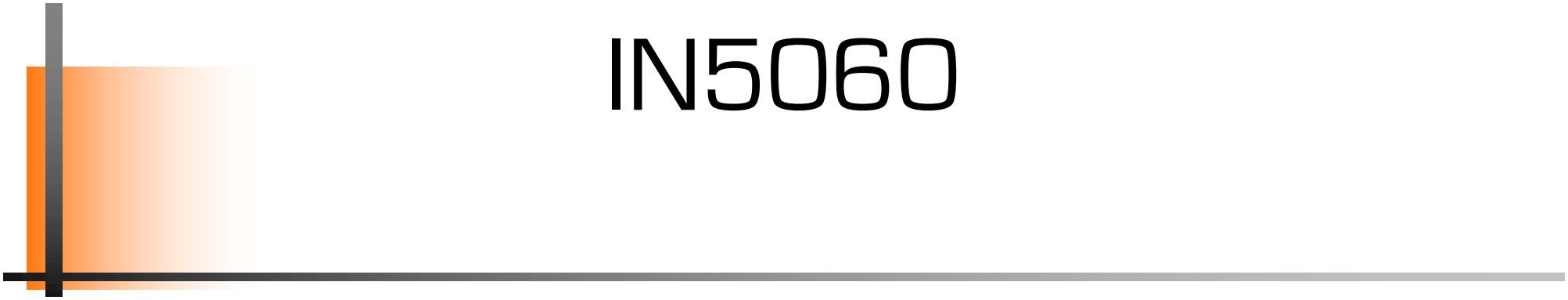


IN5060

Performance in distributed systems

Simulations

UiO : University of Oslo

Introduction

What is simulation?

Wikipedia says:

> **"Simulation** is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time."

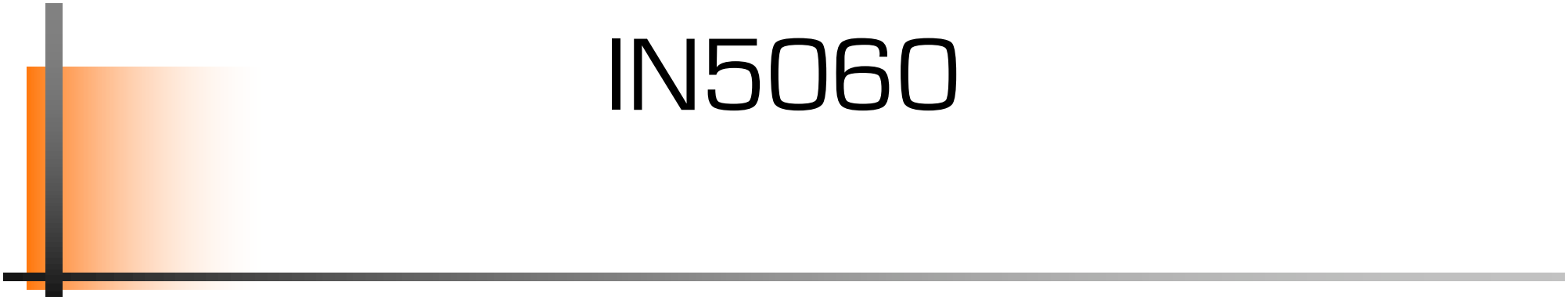
The Nature of Simulation

- Most real-world systems are too complex to allow realistic models to be evaluated analytically. These models are usually studied by means of simulation.
- **First,** see whether you can solve the problem analytically; if you cannot, then use simulation.
- **Simulation** is the technique that **imitates** the operations of a complex real-world system where a computer is generally used to evaluate a model numerically, and data are gathered in order to estimate the desired true characteristics of the model.

Introduction

- System to be characterized may not be available
  - During design or procurement stage
- Still want to predict performance
- Or, may have system but want to evaluate a wider range of workloads

  → Simulation
- However, simulations may fail
  - Need good programming, statistical analysis and performance evaluation knowledge



IN5060

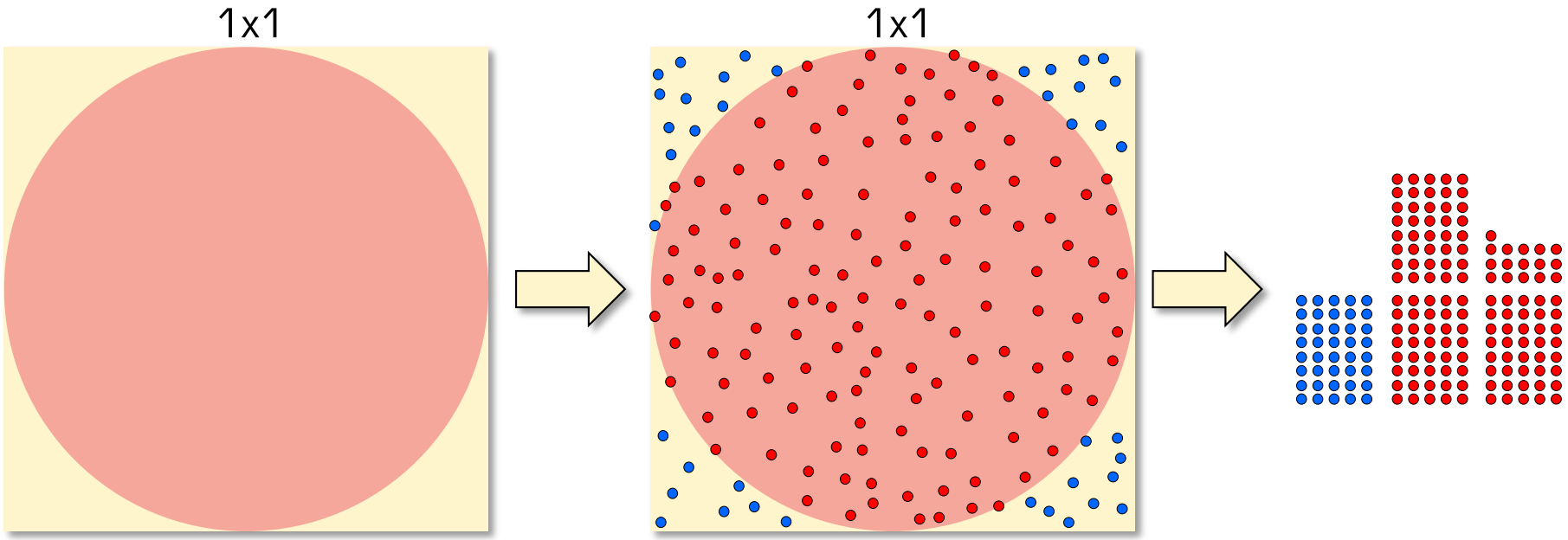
Approaches to simulation

UiO : University of Oslo

Terminology

Static and **dynamic models**

- Time is not a variable: *static*

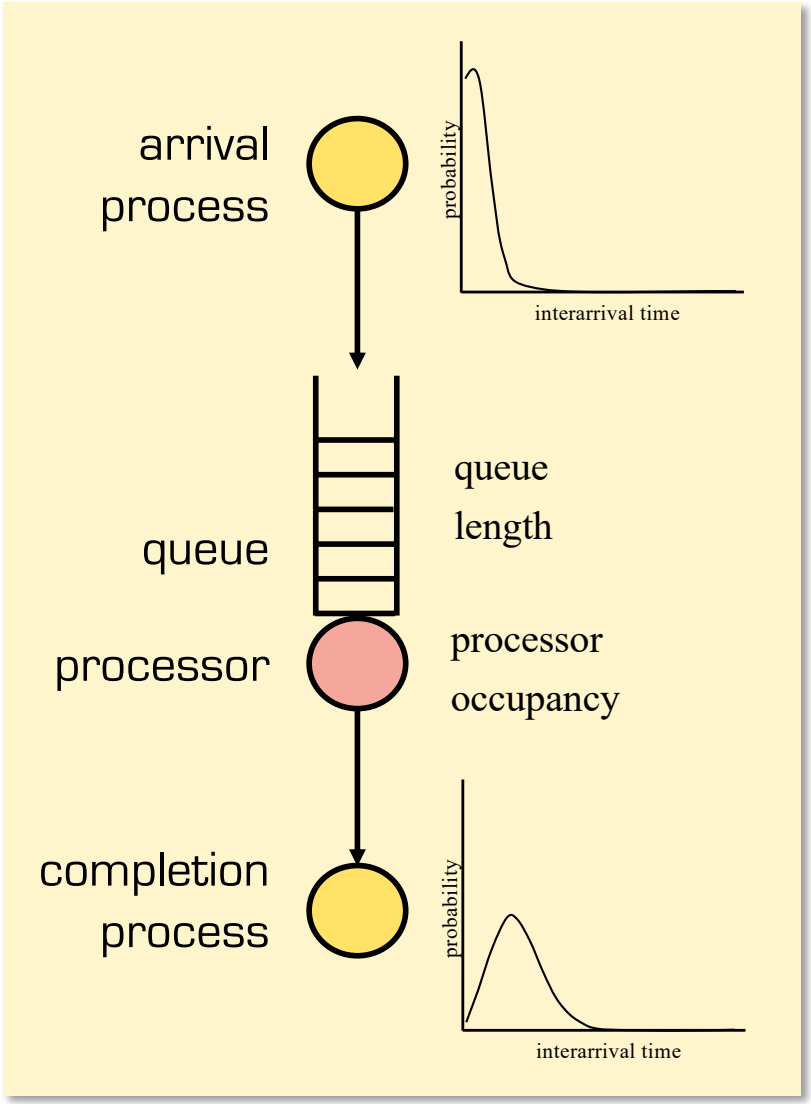


static simulation to assess the volume of a circle:

real: $\pi \frac{1}{4} = 0.785$ this static simulation: $\frac{136}{176} = 0.773$

Terminology

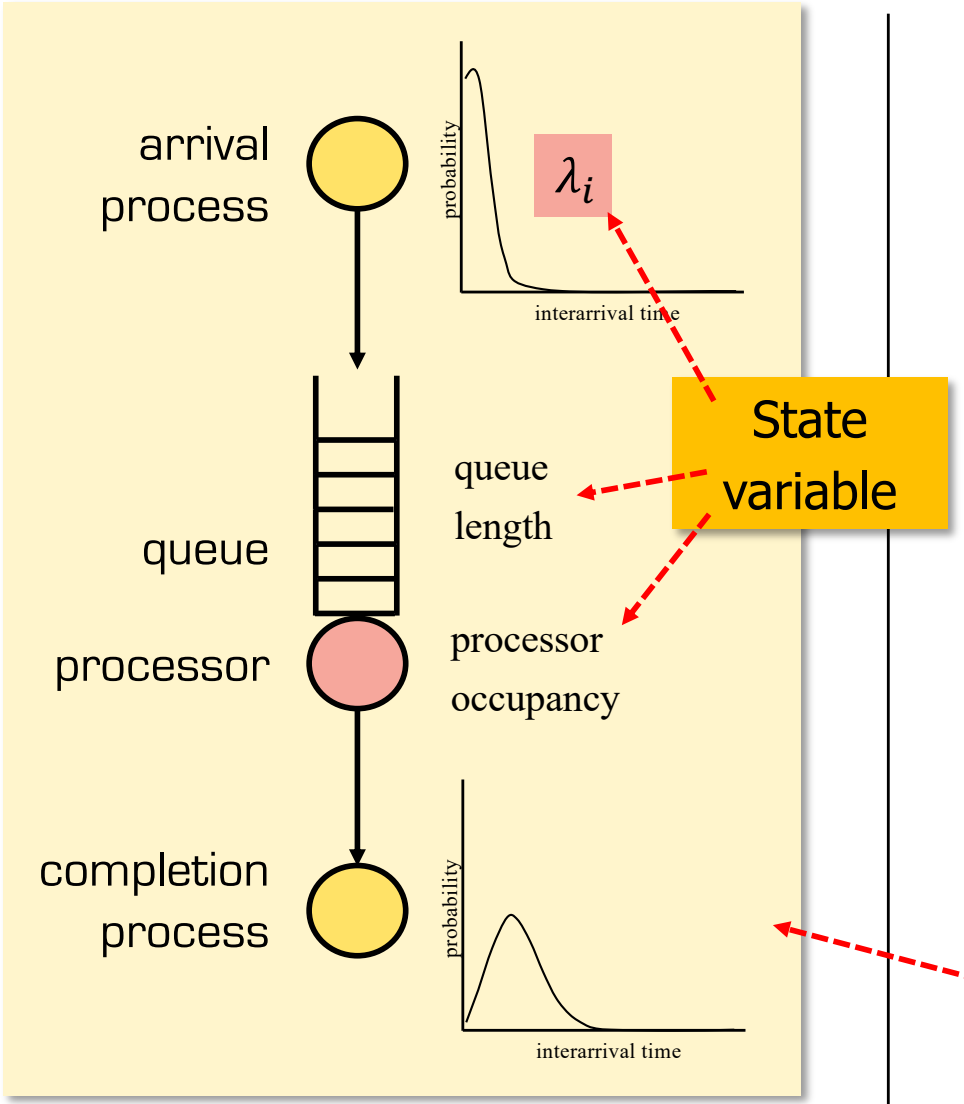
Static and **dynamic models**



- State changes with time: *dynamic*

Terminology

State and **state variables**

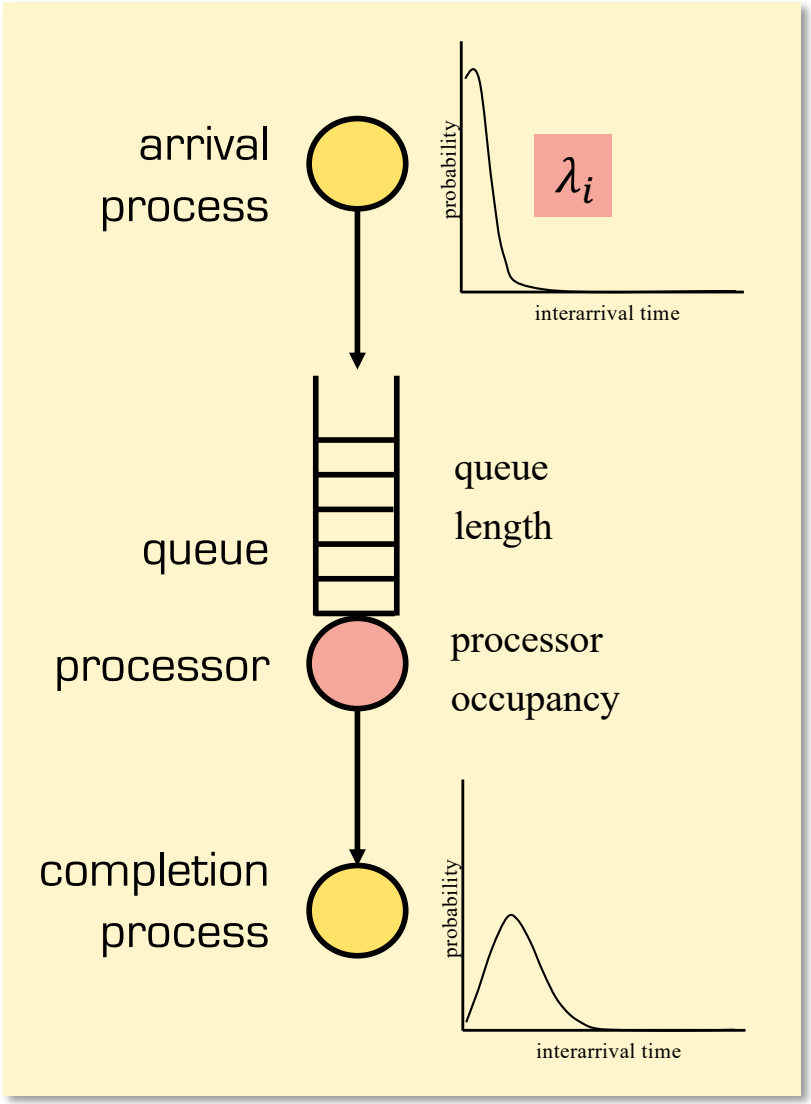


system snapshot

State

Terminology

State and **state variables**



Systems can be simulated at a very high level of abstraction

- artificial workloads
- simplified work characteristics
- removal of most system details
- focus on a particular component

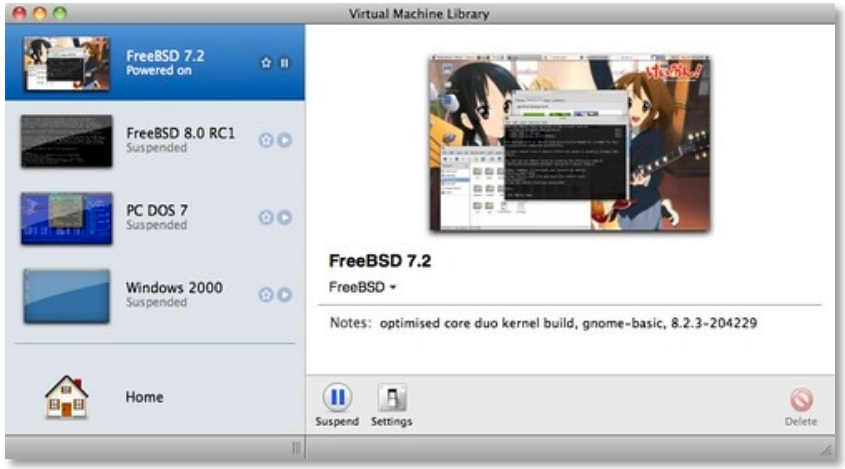
Terminology

State and **state variables**

Systems can be simulated with a large amount of detail

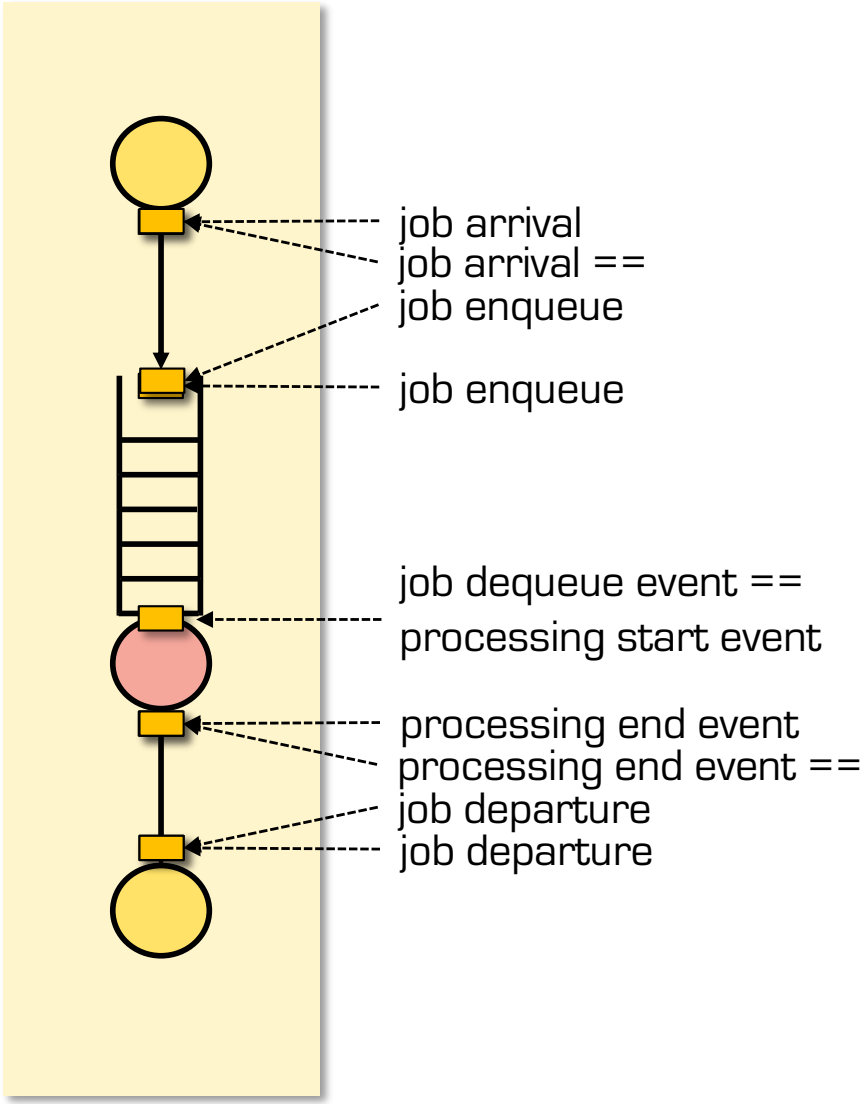
- replaying traces of real (measured or logged) workloads
- using components of real-world systems
- with fine-grained monitoring
- in controlled environments

system snapshot



Terminology

Event

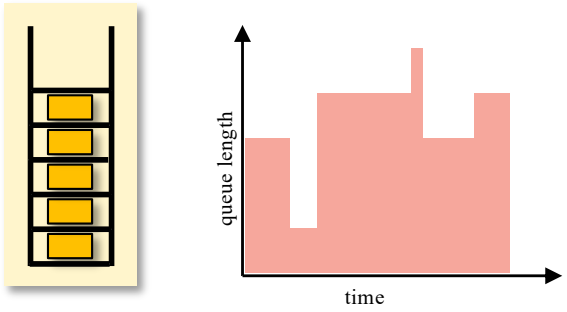


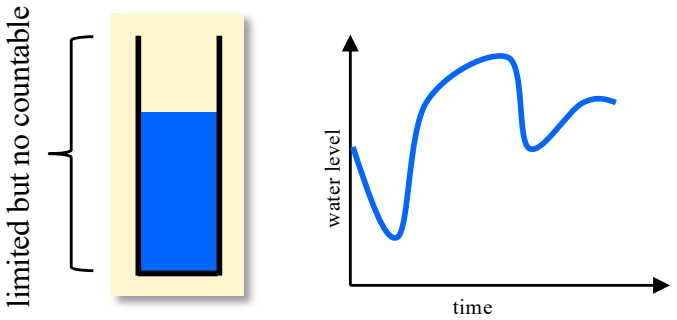
- A change in system state
- Easily explained when *state* is *discrete*
- Examples:
  - arrival of job
  - beginning of new execution
  - departure of job

Terminology

Continuous-state or **discrete-state models**

- Discrete state
  - State variables have a *countable* and *finite* or *infinite* number of states
- Continuous state
  - State variables have an *uncountable* number of states from a *finite* or *infinite* range



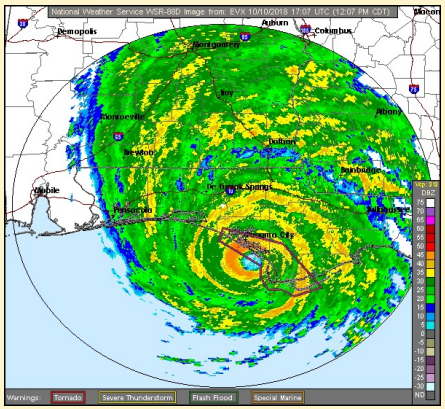


Note: *conceptually* uncountable and infinite: computer nature implies all is countable and finite

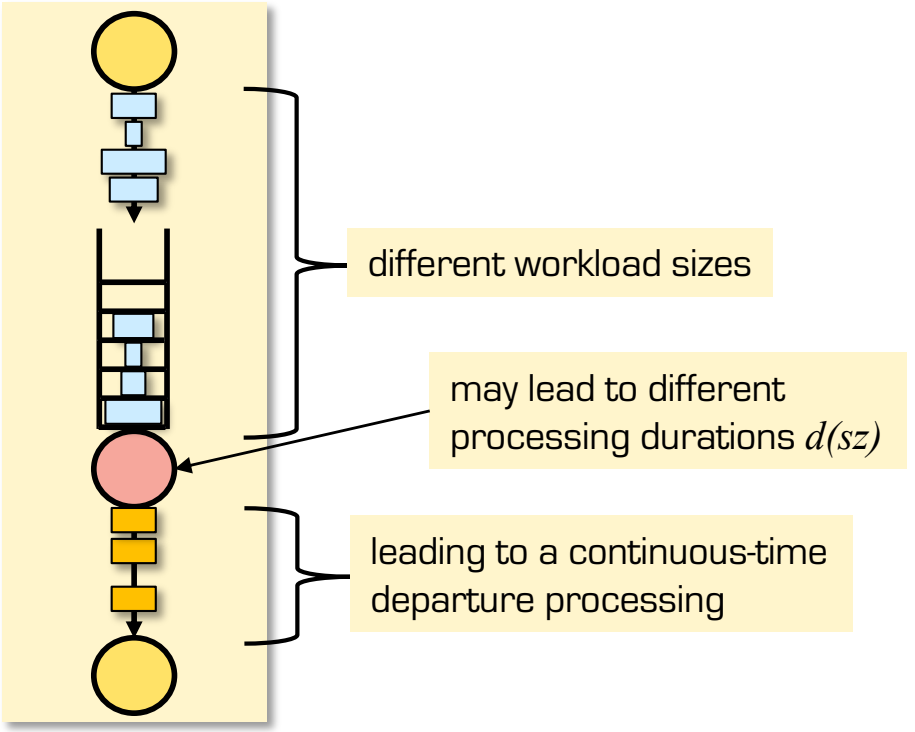
Terminology

Continuous-time or **discrete-time models**

- Discrete time
  - State is only defined at certain instances of time
- Continuous time
  - State is defined at all times



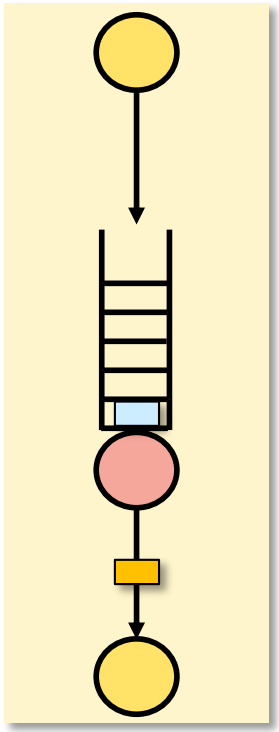
especially useful when the state space is very large

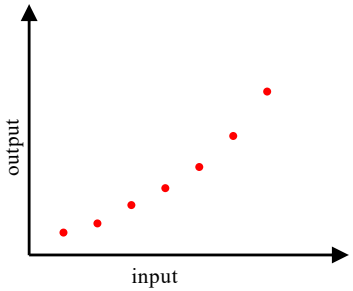


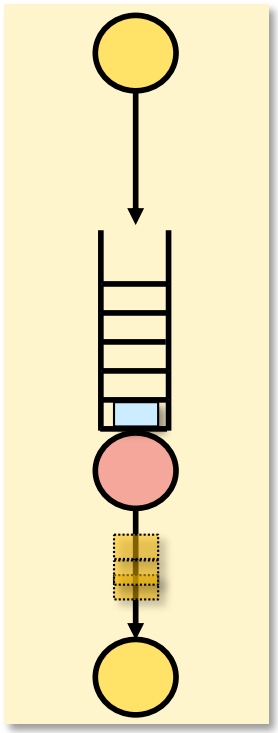
Terminology

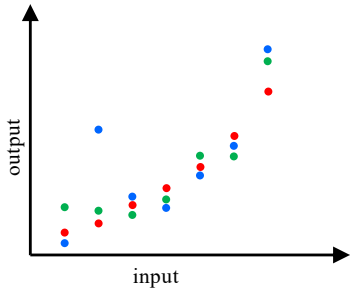
Deterministic or **probabilistic models**

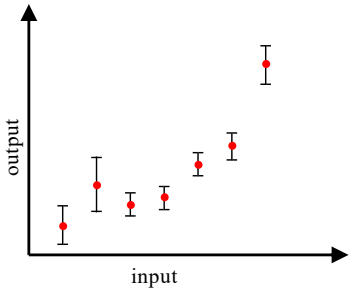
- If output predicted with certainty → *deterministic*
- If output different for different repetitions → *probabilistic*







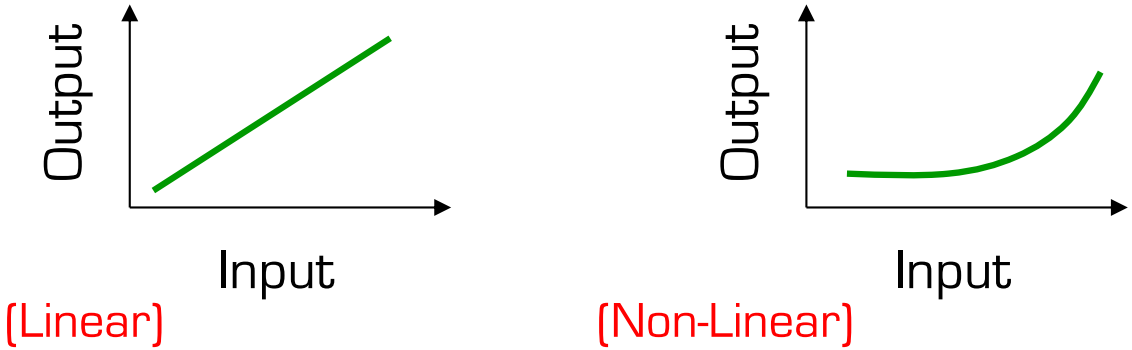




Terminology

Linear or **non-linear models**

- Output is linear combination of input → *linear*
- Otherwise → nonlinear

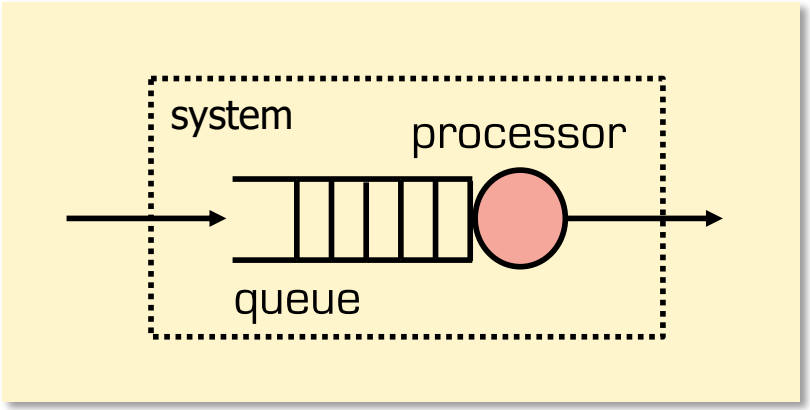


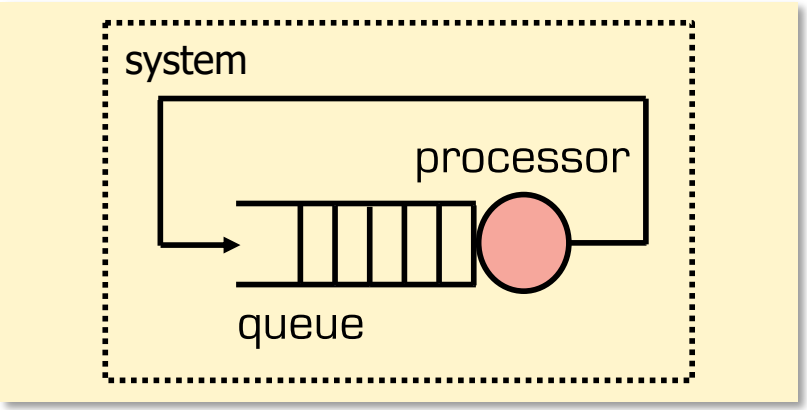
- Systems that are known to be linear can frequently be handled by analytical studies

Terminology

- *Open and closed models*
  - input is external and independent → *open*
  - model has no external input → *closed*

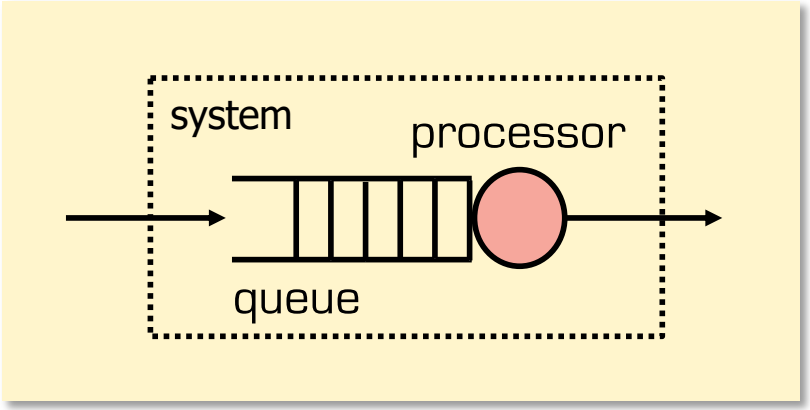
  - If same jobs leave and re-enter queue then closed, while if new jobs enter system then open

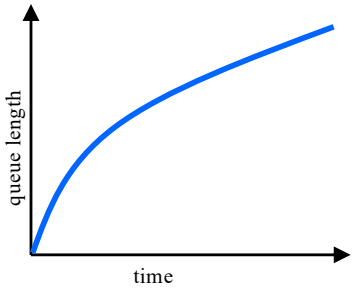




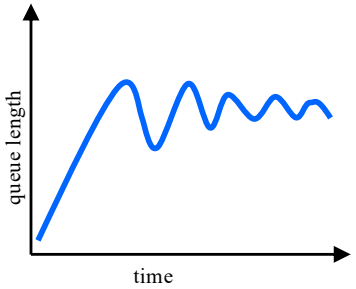
Terminology

- *Stable and unstable*
  - Model output settles down → *stable*
  - Model output always changes → *unstable*

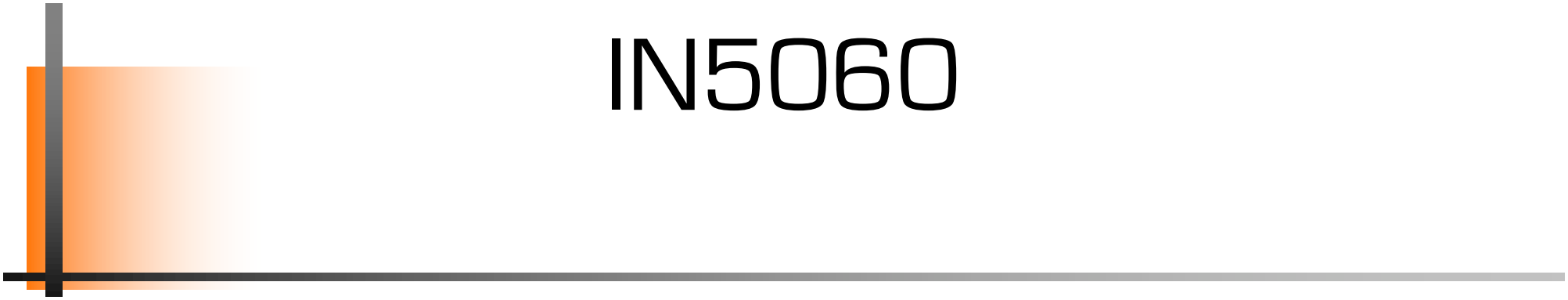




unstable



stable



IN5060

Simulation platforms

UiO : University of Oslo

Simulation Platforms

Simulation language

General-purpose language

Extended general-purpose language

Simulation package

Historical concept

- languages dedicated to simulation are quite outdated
- but they have strongly inspired general purpose languages

- GPSS (General Purpose Simulation System, 1960)
  → CSMP III (Continuous System Modelling Program)
  → APL (A Programming Language, 1966)
  → Matlab (1984)

- Simula (1962)
  → object-oriented programming in general, and
  → the Beta language in particular

Simulation Platforms

Simulation language

General-purpose language

Extended general-purpose language

Simulation package

Frequently used

- in its pure form only for very small simulations, or
- to achieve extreme performance

Non-specific libraries fall into this category

- MPI-2 for communication in high-performance computing is mostly used for very large-scale simulations

The borderline between GP and Extended GP is very fuzzy

Simulation Platforms

Simulation language

General-purpose language

Extended general-purpose language

Simulation package

Comes in many forms

- language extensions dedicated to simulation
  - rare (e.g. extensions to SysML for simulation in 2017)
- libraries dedicated to simulation
  - SIM.JS
  - SimPy
  - SystemC
- tightly integrated scripting and general purpose language
  - ns-2 (Tcl/Tk + C + library)
  - ns-3 (Python + C++ + library)
  - OMNeT++ (NED + C++ + library)

Simulation Platforms

Simulation language

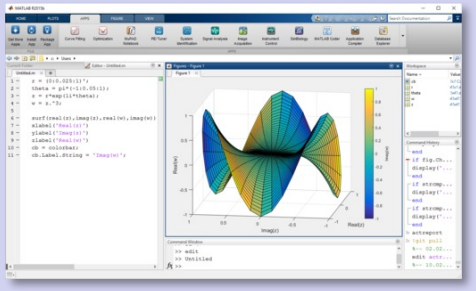
General-purpose language

Extended general-purpose language

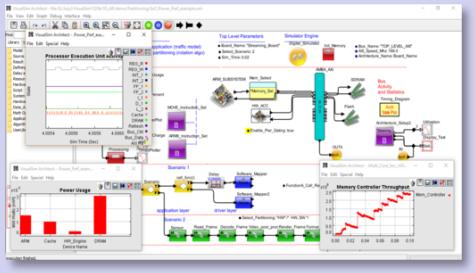
Simulation package

Very usual outside of discrete-state modeling

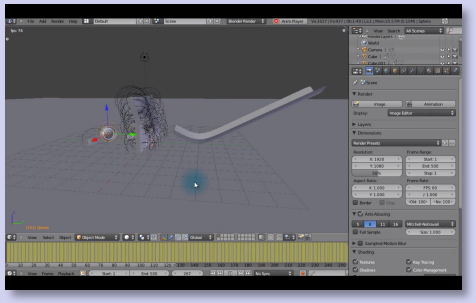
- Matlab and Octave



- VisualSim



- Blender



- many more

Selecting a Simulation Language

- Tradeoffs
  - Cost and flexibility
    - simulation languages require startup time to learn
    - general purpose language extensions require startup time to learn
    - general purpose languages may require a lot of code writing
    - packages may be feature-rich, allow visual presentation without overhead, allow to do simple simulations quickly
    - extending packages for special needs may be very hard

Types of Simulations

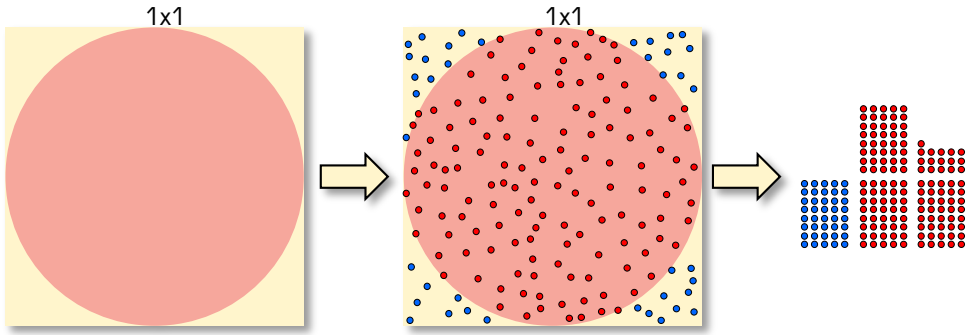
For people in networking, operating systems, distributed systems, the main types of simulation are:

- Monte Carlo simulation
- trace-drive simulation
- discrete-event simulation
- emulation

Monte Carlo Simulation

- A static simulation that has no time parameter
  - Runs until some equilibrium state reached
- Used to model physical phenomena, evaluate probabilistic system, numerically estimate complex mathematical expressions
- Driven with random number generator
  - name “Monte Carlo” comes from the random draws in casinos

Monte Carlo Simulation



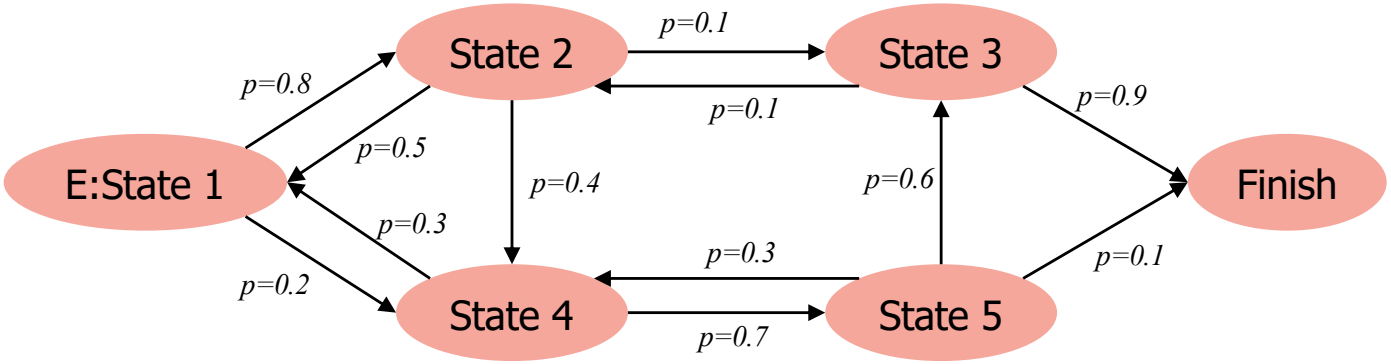
static simulation to assess the volume of a circle:

real: $\pi \frac{1}{4} = 0.785$ this static simulation: $\frac{136}{176} = 0.773$

- Make random draws from a pool of random numbers
  - here: (x,y), where x and y are randomly drawn from $[0..1]$
  - determine if (x,y) is inside the circle: $\sqrt{x^2 + y^2} \leq \frac{1}{2}$
  - count the ratio of inliers vs outliers
  - since the square surface area is 1, counting achieves the fraction inside the circle
  - draw random numbers until the accuracy is satisfactory

Monte Carlo Simulation

- Markov-Chain Monte-Carlo simulation



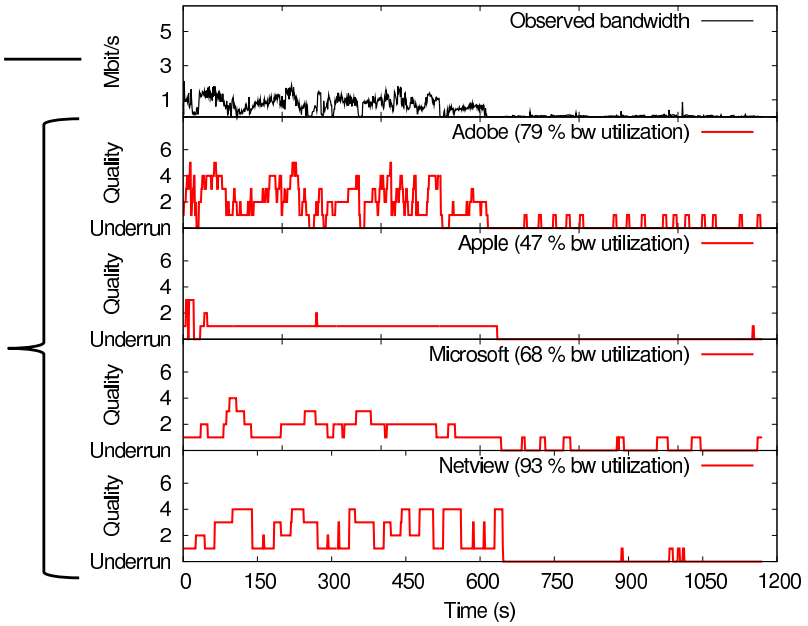
- Markov Chain: for each state, probability ranges are assigned to alternative transitions to new states (usually also *keep state*)
- in each round, a random number is drawn
- the appropriate transition is taken
- the movement through the state space is called a *random walk*
- if the system converges,
  probabilities of being in State N can be computed by repeated Monte-Carlo simulations of complete runs
- **Note:** for converging simple models, a mathematical solutions exist

Trace-Driven Simulation

- Uses time-ordered record of events on real system as input

trace

4 simulated behaviours based on the trace



- Note: need trace to be independent of system under test
  - This is very frequently forgotten !
  - For example, arrival rate of packets in TCP depends on packet loss and RTT and cannot be simulated based on a recorded IP packet trace!

Trace-Driven Simulation Advantages

Advantage	explanation
Credibility	easier to sell than random inputs
Easy validation	when gathering trace, often get performance stats and can validate with those
Accurate workload	preserves correlation of events, don't need to simplify as for workload model
Less randomness	input is deterministic, so output may be (or will at least have less non-determinism)
Fair comparison	allows comparison of alternatives under the same input stream
Similarity to actual implementation	often simulated system needs to be similar to real one so can get accurate idea of how complex

Trace-Driven Simulation Disadvantages

Disadvantage	explanation
Complexity	requires more detailed implementation
Representativeness	trace from one system may not represent all traces
Finiteness	can be long, so often limited by space but then the recorded timespan may not be representative
Single point of validation	need to be careful that validation of performance gathered during a trace represents only 1 case
Trade-off	it is difficult to change workload since cannot change trace; changing trace would first need workload model

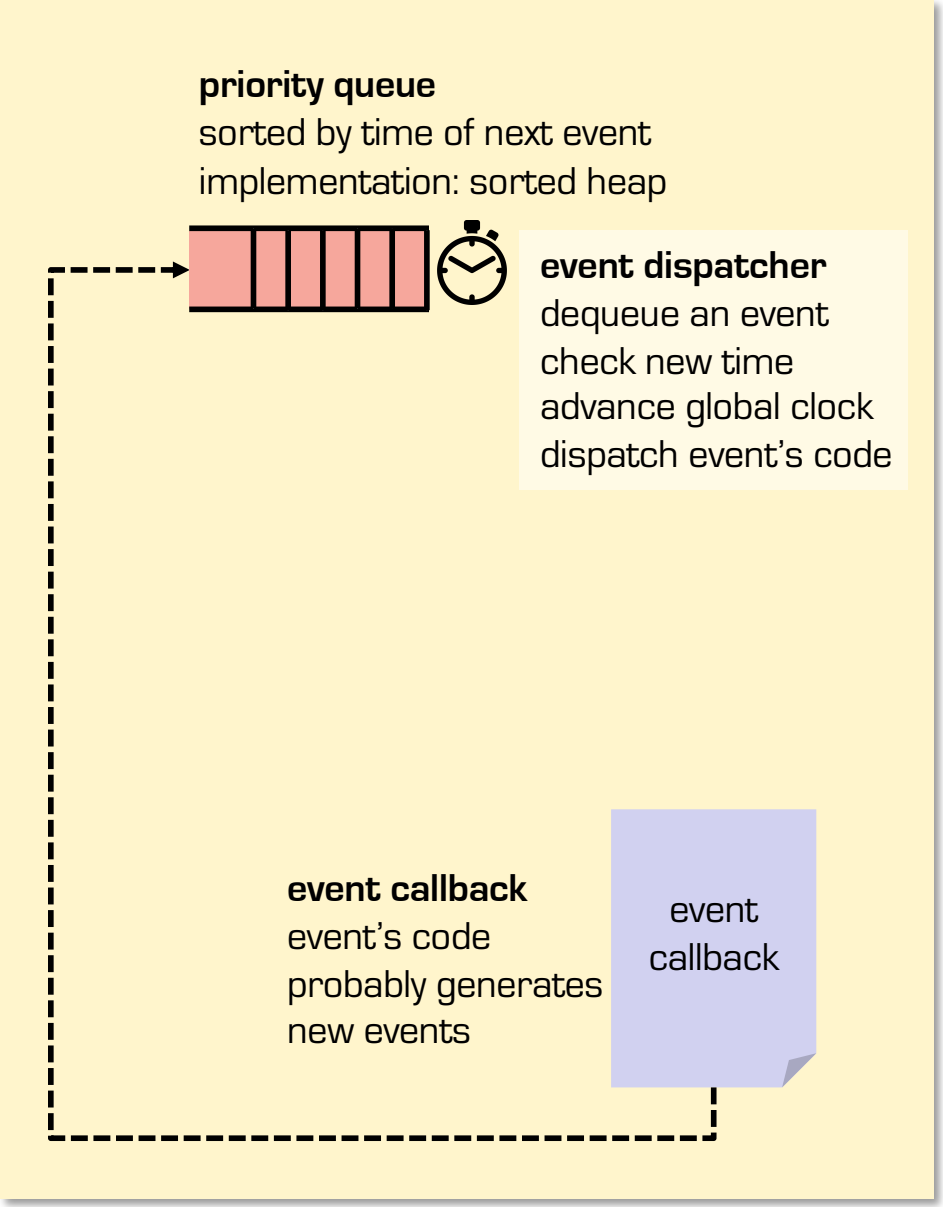
Discrete-Event Simulations

- Continuous-state simulations are highly important in many fields:
  - weather forecasting
  - chemical processes
  - geology
  - aerodynamics

  } Time is usually discrete; State is usually continuous

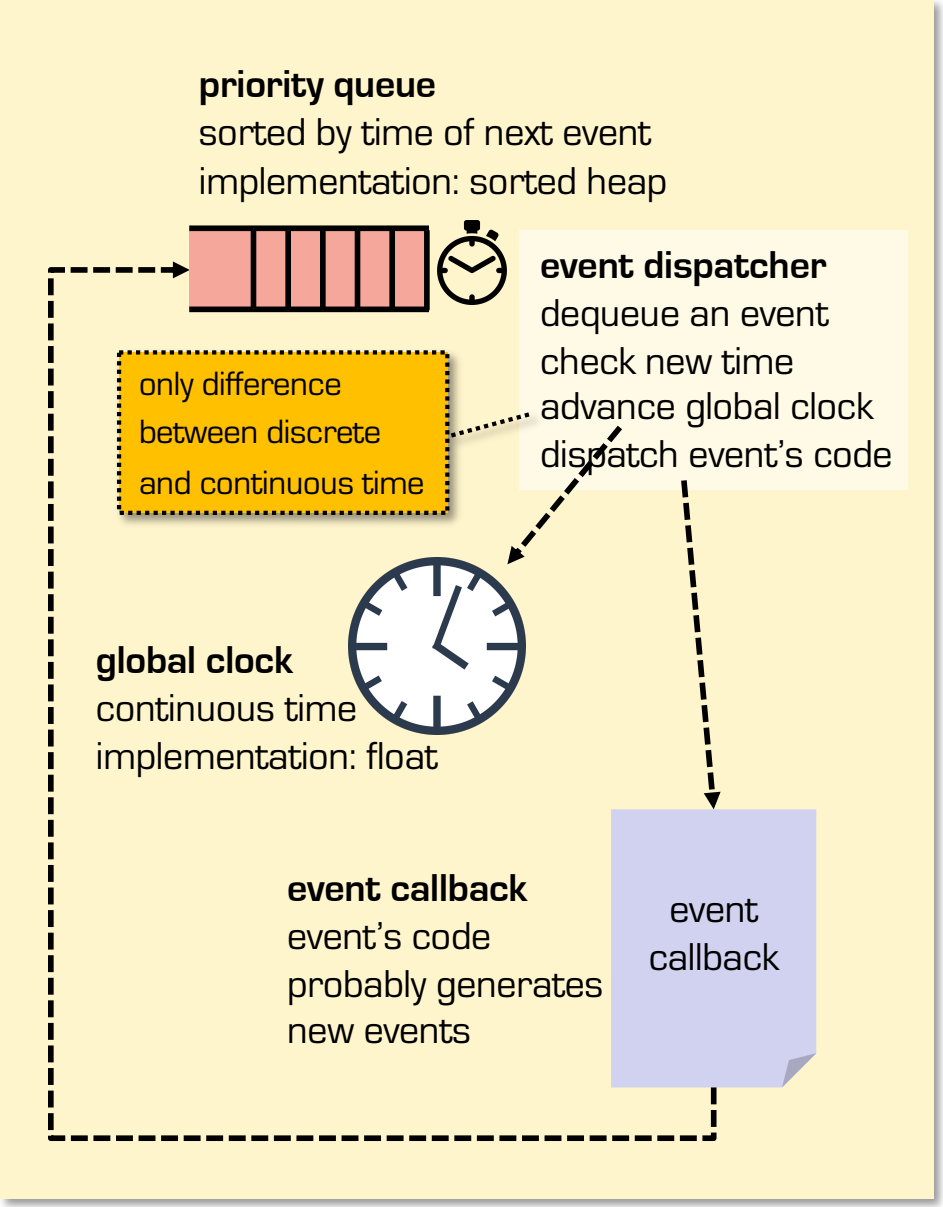
- State of computers is frequently discrete
  - packets are discrete units and packets have limited size that is countable in bytes
  - processes are scheduled in time slices, often providing sufficient resolution for simulations
  - clients' requests are countable
  - the size of memory (cache, RAM, disk, ...) is limited and countable
  - ...

Discrete-Event Simulations



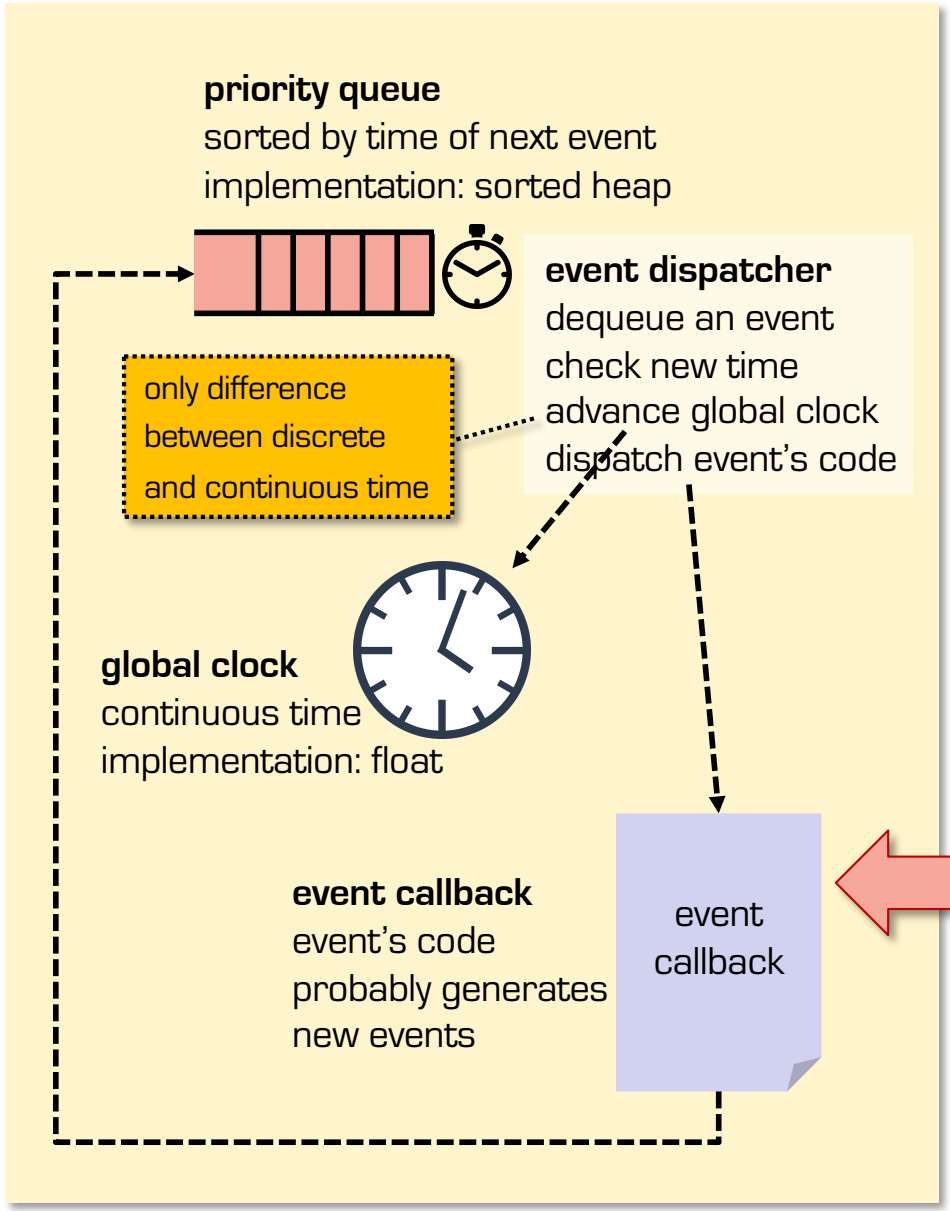
- *Event scheduler*
  - event callback generates an event and schedules it for time T
  - event is inserted into priority queue
  - callback ends, scheduler runs
  - scheduler retrieves events Y with smallest time T from priority queue
  - scheduler updates global clock to T
  - scheduler calls event dispatcher for event Y
- Event schedulers are executed very often
  - do not use an inefficient priority queue to implement the scheduler

Discrete-Event Simulations



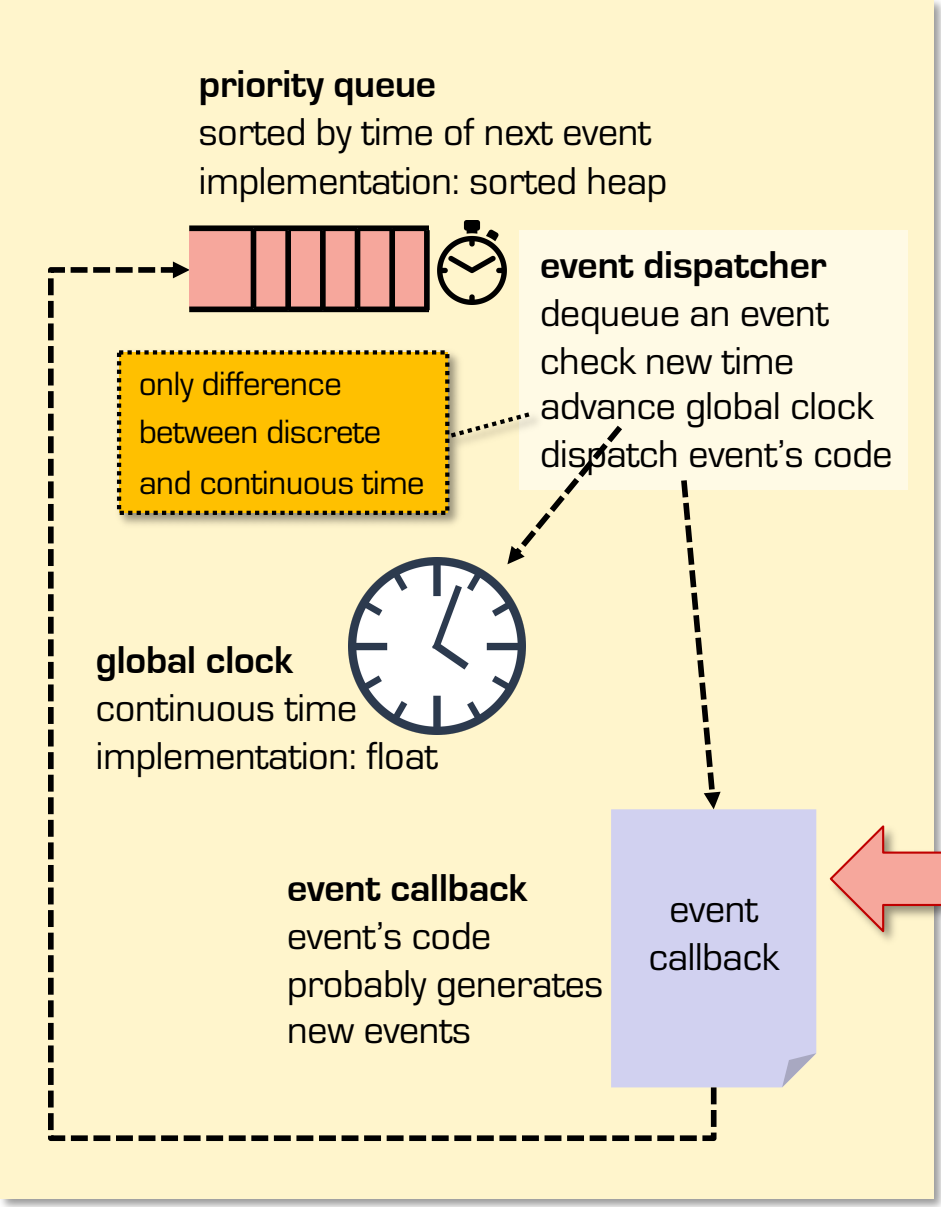
- *Simulation clock and time advancing*
  - Global variable with time
  - Scheduler advances time
    - *continuous time* – increments time by small amount and see if any events
    - *discrete time* – increments time to next event and executes
- *System state variables*
  - Global variables describing state
  - Can be used to save and restore

Discrete-Event Simulations



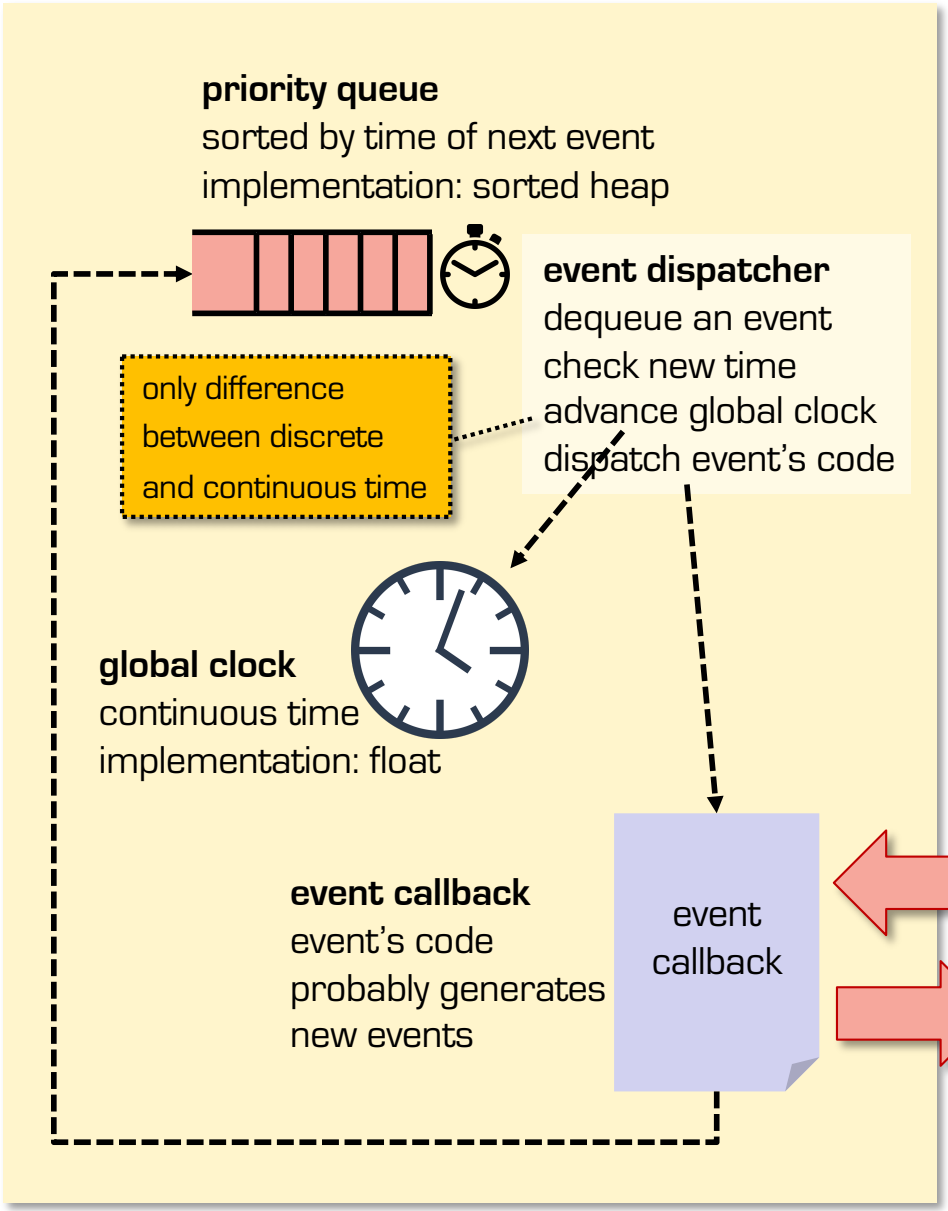
- *Event routines*
  - Specific routines to handle event
  - Often handled by callback from event scheduler
    - straightforward in object-oriented language
  - or connect to real systems for emulation

Discrete-Event Simulations



- *Input routines*
  - Get input from user (or config file, script or GUI)
  - Often get all input before simulation starts
    (not true for emulation and trace-driven simulations)
  - May allow range of inputs and number or repetitions, etc.
- *Input options*
  - Traces can be connected to event callbacks or directly to the event queue
  - Event callbacks can encapsulate emulation

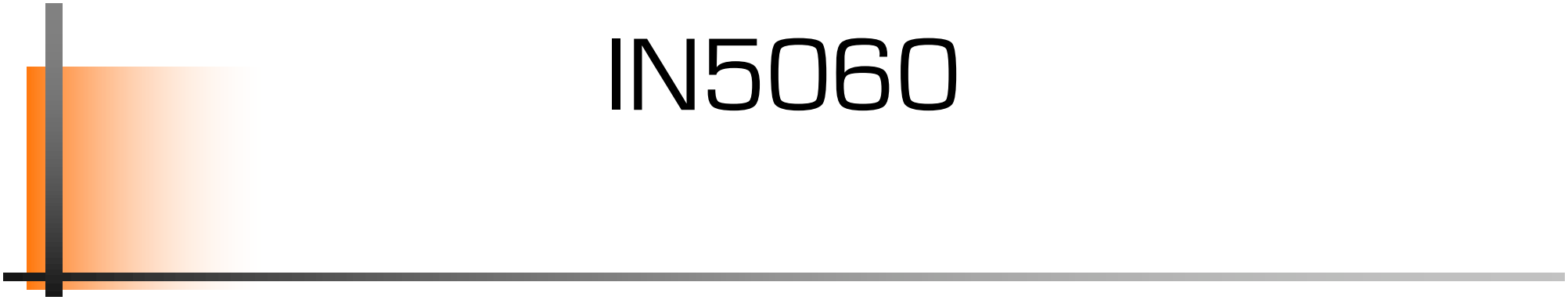
Discrete-Event Simulations



- *Report generators*
  - Routines executed at end of simulation, final result and print
  - Can include graphical representation, too
  - Ex: may compute total wait time in queue or number of processes scheduled

Questions

- Which type of simulation for each of
  - Model requester access patterns to a server where a large number of factors determine requester
  - Model scheduling in a multiprocessor with request arrivals from known distribution
  - Complex mathematical integral



IN5060

Mistakes in simulation

UiO : University of Oslo

Common Mistakes in Simulation

- *Inappropriate level of detail*
  - Level of detail often potentially unlimited
  - But more detail requires more time to develop
    - And often to run!
  - Can introduce more bugs, making more inaccurate not less!
  - Often, more detailed viewed as “better” but may not be the case
    - More detail requires more knowledge of input parameters
    - Getting input parameters wrong may lead to more inaccuracy (Ex: disk service times exponential vs. simulating sector and arm movement)
  - Start with less detail, study sensitivities and introduce detail in high impact areas

Common Mistakes in Simulation

- *Improper language*
  - Choice of language can have significant impact on time to develop
  - Special-purpose languages can make implementation, verification and analysis easier
- *Unverified models*
  - Simulations are generally large computer programs
  - Unless special steps taken, they have bugs or errors
- *Invalid models*
  - No errors, but does not represent real system
  - Need to validate models by analysis, measurement or intuition

Ideas for verification of simulation models discussed in this lecture

Common Mistakes in Simulation

- *Improperly handled initial conditions*
  - Often, initial trajectory are not representative of steady state
    - Including initial data can lead to inaccurate results
  - Typically want to discard, but need method to do so effectively
  - Discussed in this lecture
  - However:
    - the goal of a study may be the exploration of the initial behaviour of a system

- *Too short simulation runs*
  - Attempt to save time
  - Makes even *more* dependent upon initial conditions
  - Correct length depends upon the accuracy desired (confidence intervals)
  - Discussed in this lecture

Common Mistakes in Simulation

- *Poor random number generators and seeds*
  - "Home grown" are often not random enough
    - Makes artifacts
  - Best to use well-known one
  - Choose seeds that are different (Jain chapter 26)
  - since Jain's book:
    - operating systems' random number generators were proven to be poor, simplifying man-in-the-middle attacks on TCP connections
    - since then, operating systems have been equipped with much better generators (Linux: `/dev/random`, `/dev/urandom`)
    - still, simulators need a lot of random numbers and OS generators can be saturated

Common Mistakes in Simulation

Mistake	Relevance
Inappropriate level of detail	high
Improper language	costs mostly time
Unverified models	high
Invalid models	high
Improperly handled initial conditions	problematic combined with next
Too short simulation runs	high if not discovered in analysis, or simulation is not repeatable
Poor random number generators and seeds	problem has changed since Jain's book

More Causes of Failure

Any given program, when running, is obsolete. If a program is useful, it will have to be changed. Program complexity grows until it exceeds the capacity of the programmer who must maintain it. - Datamation 1968

- *Large software*
  - Quotations above apply to software development projects, including simulations
  - If large simulation efforts are not managed properly, they can fail
- *Inadequate time estimate*
  - Need time for *validation* and *verification*
  - Time needed can often grow as more details added

More Causes of Failure

- *No achievable goal*
  - Common example is “model X”
    - But there are many levels of detail for X
  - Goals: Specific, Measurable, Achievable, Repeatable, Time-bound (SMART)
  - Project without goals continues indefinitely

- *Incomplete mix of essential skills*
  - Team needs one or more individuals with certain skills
  - Need: leadership, modeling and statistics, programming, knowledge of modeled system

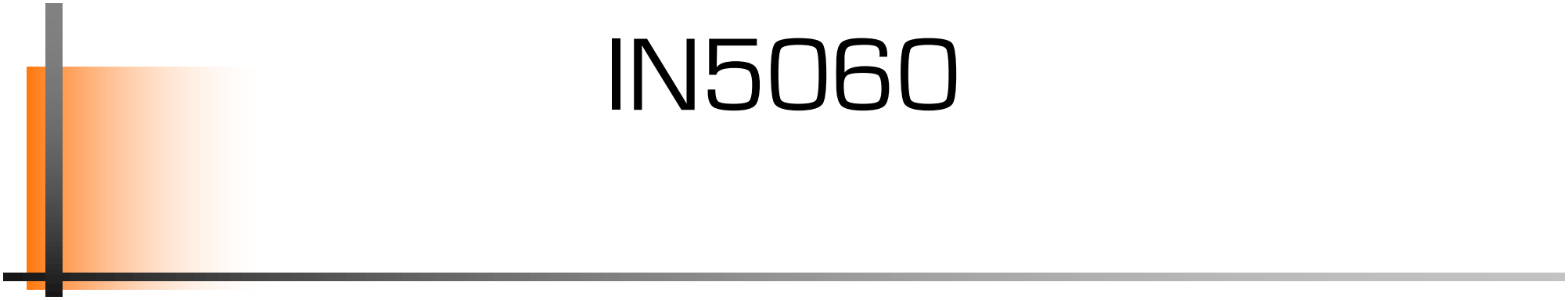
Simulation Checklist

- Checks before developing simulation
  - Is the goal properly specified?
  - Is detail in model appropriate for goal?
  - Does team include right mix (leadership, modeling, programming, background)?
  - Has sufficient time been planned?

- Checks during simulation development
  - Is random number random?
  - Is model reviewed regularly?
  - Is model documented?

Simulation Checklist

- Checks after simulation is running
  - Is simulation length appropriate?
  - Are initial transients removed?
  - Has model been verified?
  - Has model been validated?
  - Are there any surprising results?
    - If yes, have they been validated?



IN5060

Analysis of results (verification)

UiO : University of Oslo

Analysis of Simulation Results

> *Always assume that your assumption is invalid.*
> – Robert F. Tatman

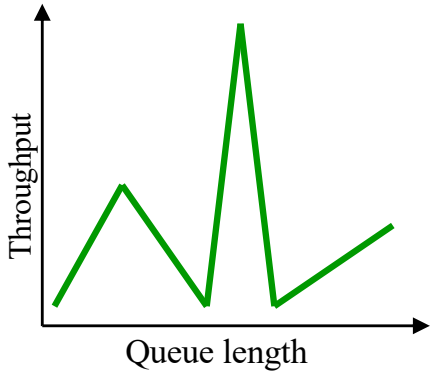
- Would like model output to be close to that of real system
- Made assumptions about behavior of real systems
- 1st step, test if assumptions are reasonable
  - *Validation*, or representativeness of assumptions
- 2nd step, test whether model implements assumptions
  - *Verification*, or *correctness*

Model Verification Techniques

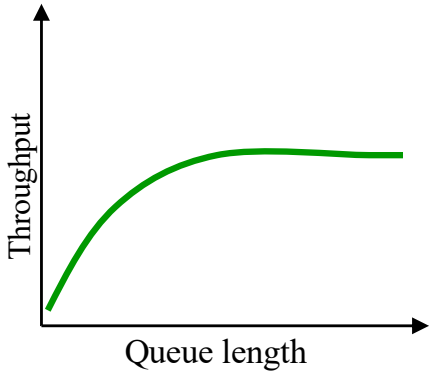
- Good software engineering practices will result in fewer bugs
- Top-down, modular design
- Assertions (antibugging)
  - Say, $total\ packets = packets\ sent + packets\ received$
  - If not, can halt or warn
- Structured walk-through
- Simplified, deterministic cases
  - Even if a simulation is complicated and non-deterministic, use simple repeatable values (maybe fixed seeds) to debug
    - do not use `/dev/random` or `/dev/urandom` as random number source while debugging, use a random number generator that is thread-specific with a user-adjustable seed
- Tracing
  - via print statements or debugger

Model Verification Techniques

- Continuity tests
  - Slight change in input should yield slight change in output, otherwise error



non-sense result
probably simulation error



expected result
no indication for error

- Degeneracy tests
  - Try known extremes (e.g. lowest and highest) since they may reveal bugs

Model Verification Techniques

- Consistency tests – similar inputs produce similar outputs
  - Ex: 2 sources at 50 pkts/sec produce same total as 1 source at 100 pkts/sec
- Seed independence – random number generator starting value should not affect final conclusion
  - it will probably change the specific output
  - it should not change the overall conclusions
  - however:
    - this can be the real behaviour in an unstable system
    - the simulated period may be too short to reach stability

Model Validation Techniques

- Ensure assumptions used are reasonable
  - Want final simulated system to be like real systems
- Unlike verification, techniques to validate one simulation may be different from one model to another

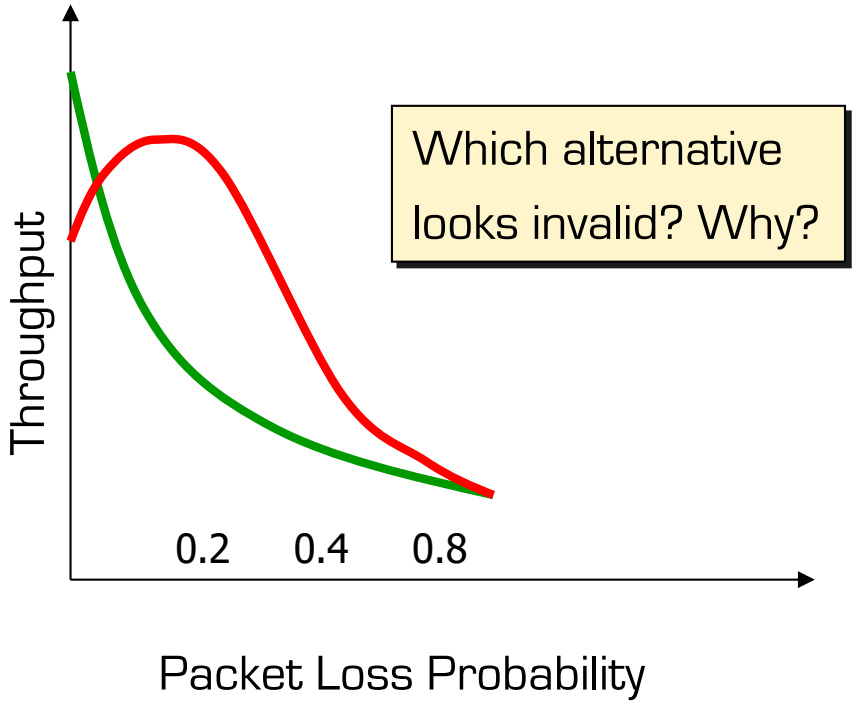
- Three key aspects to validate:
  - Assumptions
  - Input parameter values and distributions
  - Output values and conclusions

- Compare validity of each to one or more of:
  - Expert intuition
  - Real system measurements
  - Theoretical results

→ 9 combinations
Not all are always possible, however

Model Validation Techniques - Expert Intuition

- Most practical, most common
- "Brainstorm" with people knowledgeable in area
- Assumptions validated first, followed soon after by input. Output validated as soon as output is available (and verified), even if preliminary
- Present measured results and compare to simulated results (can see if experts can tell the difference)



Model Validation Techniques - Real System Measurements

- Most reliable and preferred
- May be infeasible because system does not exist or too expensive to measure
  - That could be why simulating in the first place!
- But even one or two measurements add an enormous amount to the validity of the simulation
- Should compare input values, output values, workload characterization
  - Use multiple traces for trace-driven simulations
- Can use statistical techniques (confidence intervals) to determine if simulated values different than measured values

Model Validation Techniques - Theoretical Results

- Can be used to compare a simplified system with simulated results
- May not be useful for sole validation but can be used to complement measurements or expert intuition
  - Ex: measurement validates for one processor, while analytic model validates for many processors
- Note, there is no such thing as a “fully validated” model
  - Would require too many resources and may be impossible
  - Can only show is invalid
- Instead, show validation in a few select cases, to lend *confidence* to the overall model results

IN5060

Analysis of results

initial transients

UiO : University of Oslo

Transient Removal

- Most simulations only want steady state
  - Remove initial *transient* state
  - but not always
    e.g. initial competition between TCP flows is interesting
- Trouble is, not possible to define exactly what constitutes end of transient state
- Use heuristics:
  - Long runs
  - Proper initialization
  - Truncation
  - Initial data deletion
  - Moving average of replications
  - Batch means

Long Runs

- Use very long runs
- Effects of transient state will be amortized
- But … wastes resources
- And tough to choose how long is “enough”
- Recommendation … don’t use long runs alone

Proper Initialization

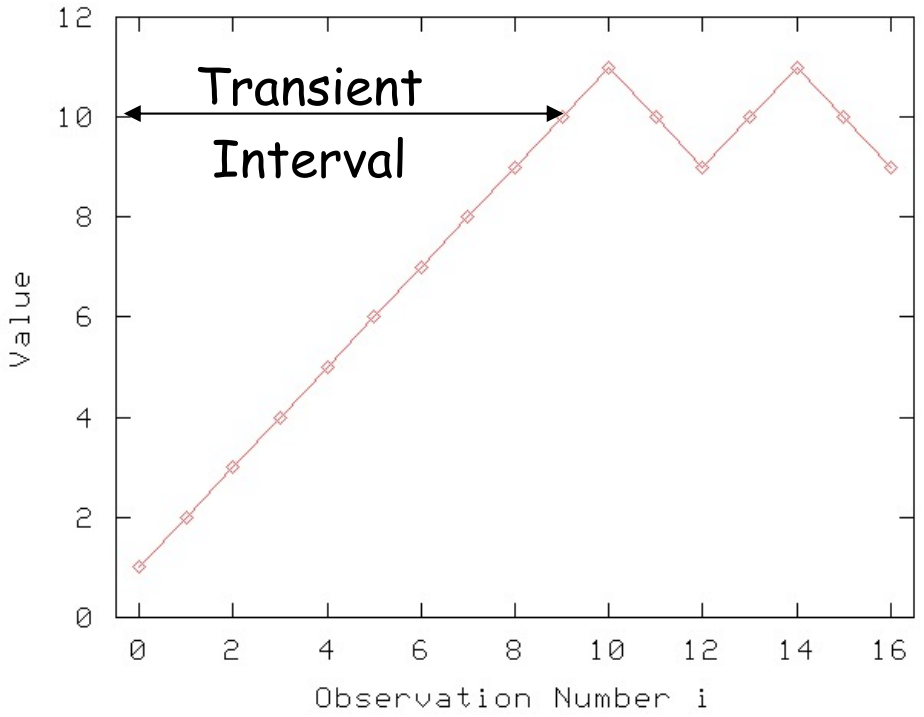
- Start simulation in state close to expected state
  - Ex: CPU scheduler may start with some jobs in the queue
- Determine starting conditions by previous simulations or simple analysis
- May result in decreased run length, but still may not provide confidence that the simulation has reached a stable condition

Truncation

- Assume variability during steady state is less than during transient state
- Variability measured in terms of range
  - (min, max)
- If a trajectory of range stabilizes, then assume that simulation is in stable state
- Method:
  - Given n observations $\{x_1, x_2, \ldots, x_n\}$
  - ignore first l observations
  - Calculate (min,max) of remaining n-l
  - Repeat for $l = 1 \ldots n$
  - Stop when observation $l+1$ is neither min nor max

Truncation Example

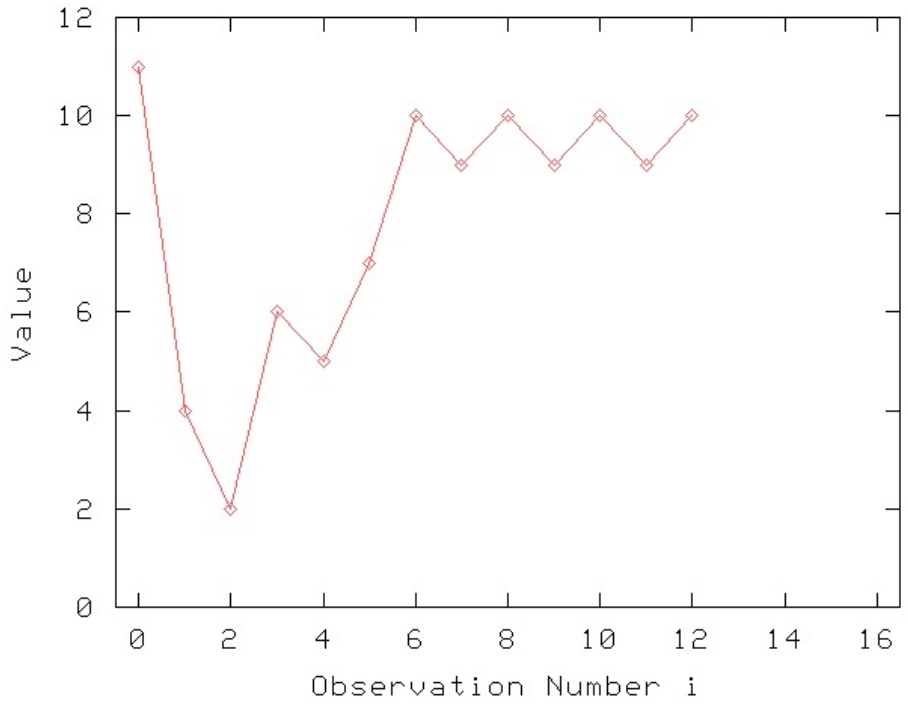
- Sequence:
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9, 10, 11, 10, 9...
- Ignore first ($l=1$),
  range is $(2, 11)$ and
  2nd observation ($l+1$) is the min
- Ignore second ($l=2$),
  range is $(3,11)$ and
  3rd observation ($l+1$) is min
- Finally, $l=9$ and
  range is $(9,11)$ and
  10th observation is
  neither min nor max
- So, discard first 9 observations



Truncation Example 2

- Find duration of transient interval for:

  11, 4, 2, 6, 5, 7, 10, 9, 10, 9, 10, 9, 10

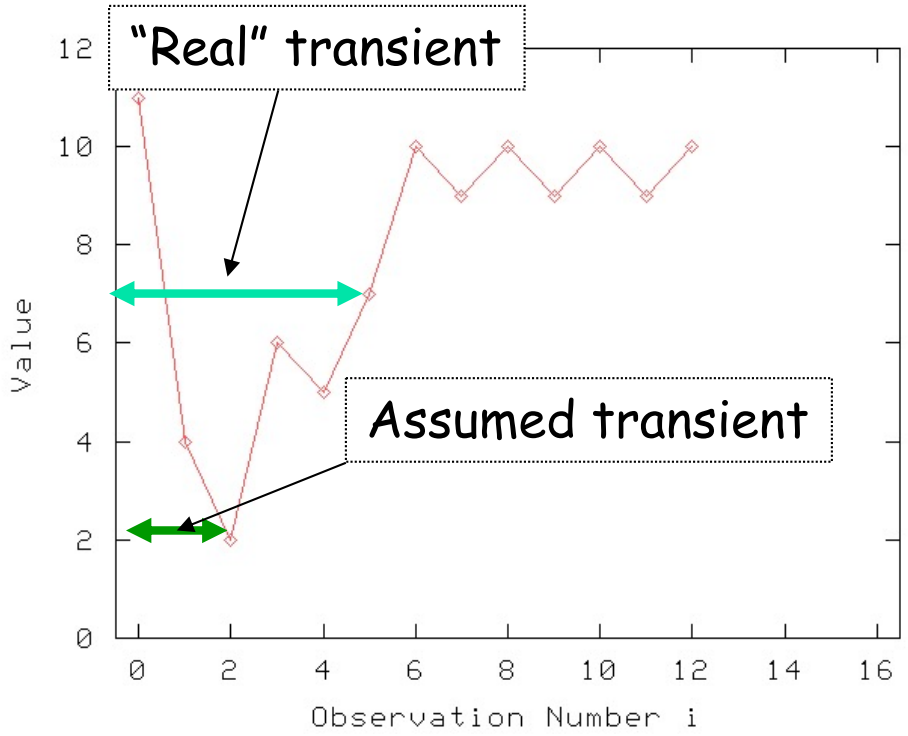


Truncation Example 2

- Find duration of transient interval for:

  11, 4, 2, 6, 5, 7, 10, 9, 10, 9, 10, 9, 10

- When l=3,
  range is *(5,10)* and
  4^{th} *(6)* is not min or max
- So, discard only 3
  instead of 6



Initial Data Deletion

- Study average after some initial observations are deleted from sample
  - If average does not change much,
    must be deleting from steady state
  - However, since randomness can cause some fluctuations during steady state,
    need multiple runs (with different seeds)

- Given m replications (m simulation runs with different seeds)
  of size n (n observations are taken)
  each with x_{ij} j-th observation of i-th replication
  - Note: j varies along time axis and i varies across replications

Initial Data Deletion

- Get mean trajectory for observation j

$$\bar{X}_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$$

- Get overall mean

$$\bar{\bar{X}} = \frac{1}{n}\sum_{j=1}^{n} \bar{X}_j$$

- Assume transient state l long, delete first l and compute mean for the rest, so for all $l = 1..n$

$$\bar{x}_l = \frac{1}{n-l}\sum_{j=l+1}^{n} \bar{X}_j$$

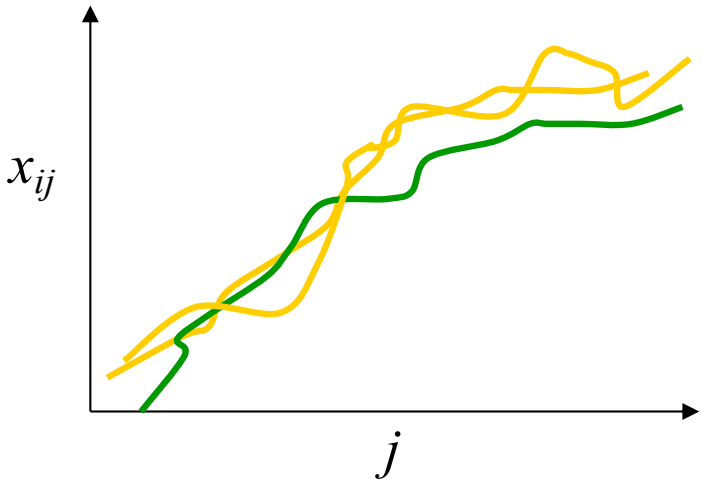
Initial Data Deletion

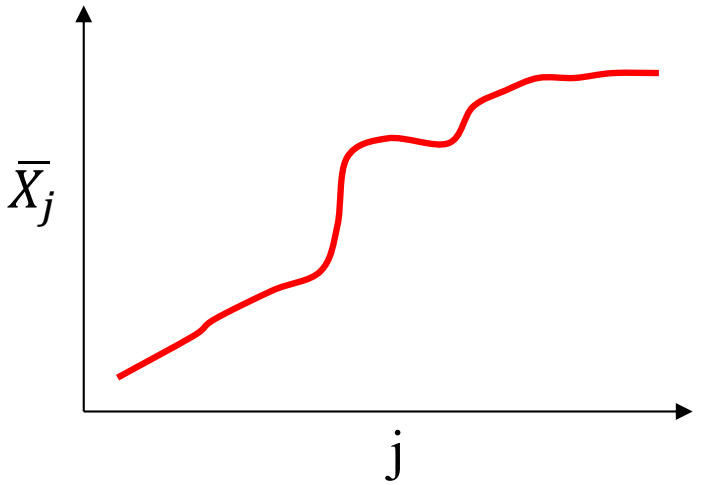
- Compute relative change for all $l = 1..n$

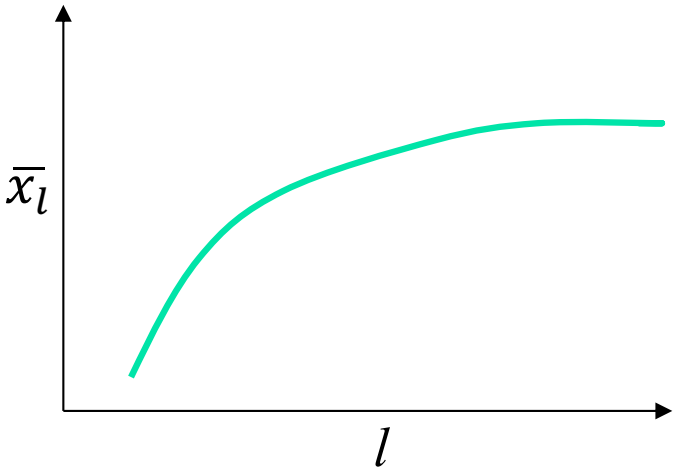
$$R_l = \frac{\bar{x_l} - \bar{X}}{\bar{X}}$$

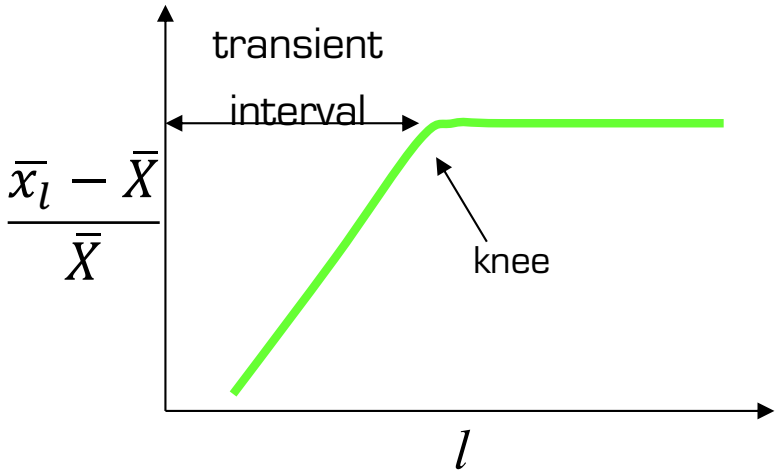
- Plot R_l over l
- Relative change graph will stabilize at knee
- Choose l there and delete 1 through l

Initial Data Deletion









Moving Average of Independent Replications

- Compute mean over moving time window
- Get mean trajectory

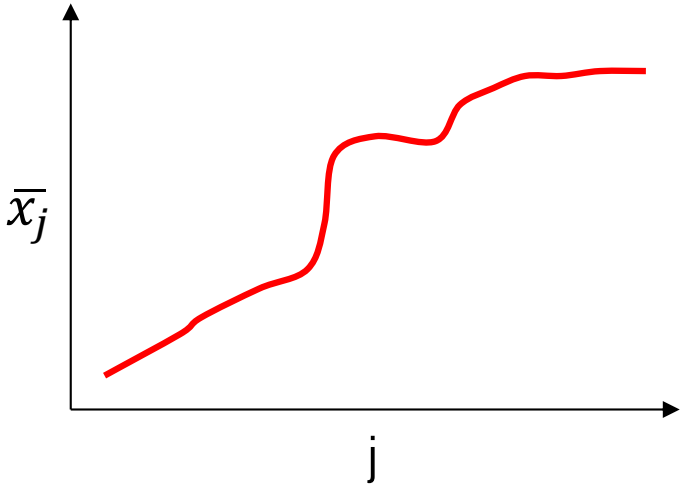
$$\overline{x_j} = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$$

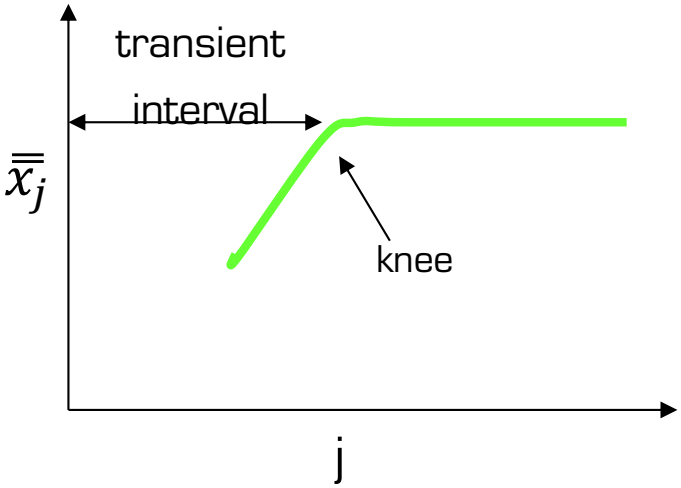
- Set $k=1$. Plot moving average of $2k+1$ values:

$$\overline{\overline{x_j}} = \frac{1}{2k+1}\sum_{l=-k}^{k} \overline{x_{j+l}}$$

for $j=k+1, k+2, \ldots, n-k$

- Repeat for $k=2,3\ldots$ and plot until smooth
- Find knee.
  Value at j is length of transient phase.





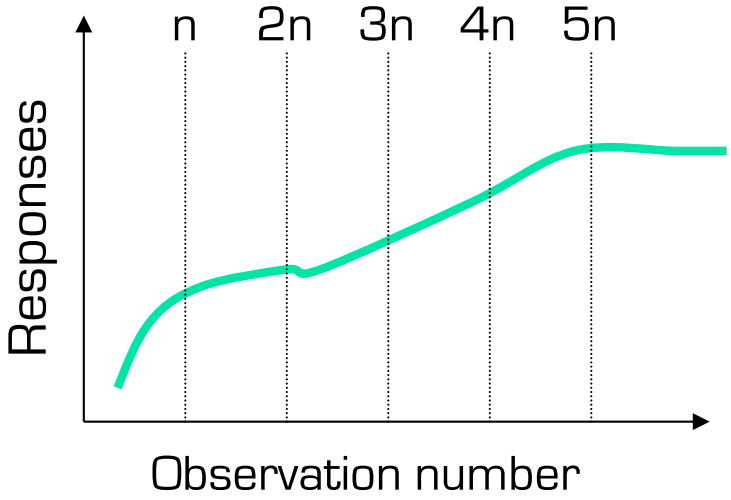
Batch Means

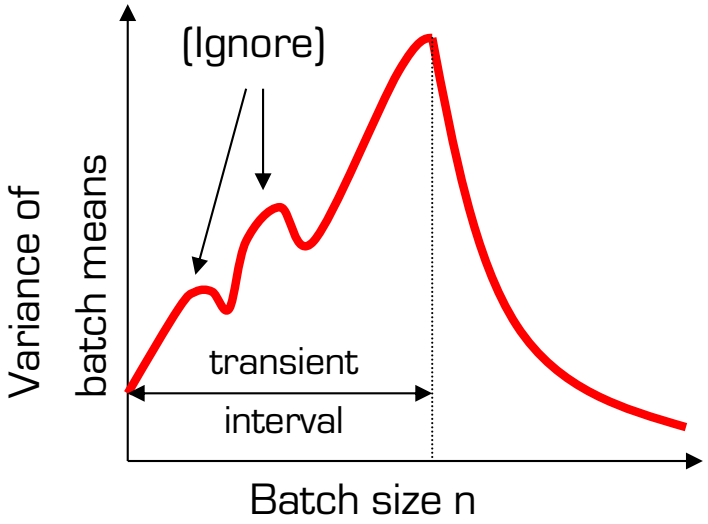
- run for long time
  - make N observations in each run
- divide up into batches
  - m batches size n each, so that $\mathrm{m} = \frac{N}{n}$
- compute batch means

$$\overline{x_i} = \frac{1}{n} \sum_{j=im+1}^{im+n} x_j$$

- and overall mean

$$\overline{\overline{x}} = \frac{1}{m} \sum_{i=1}^{m} \overline{x_i}$$





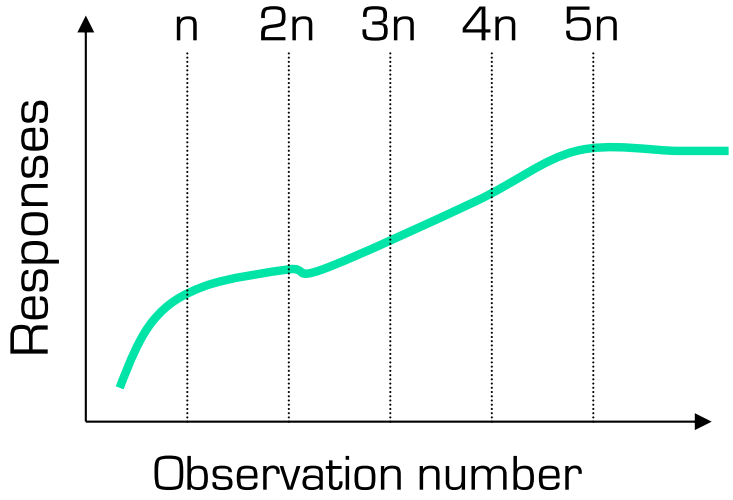
Batch Means

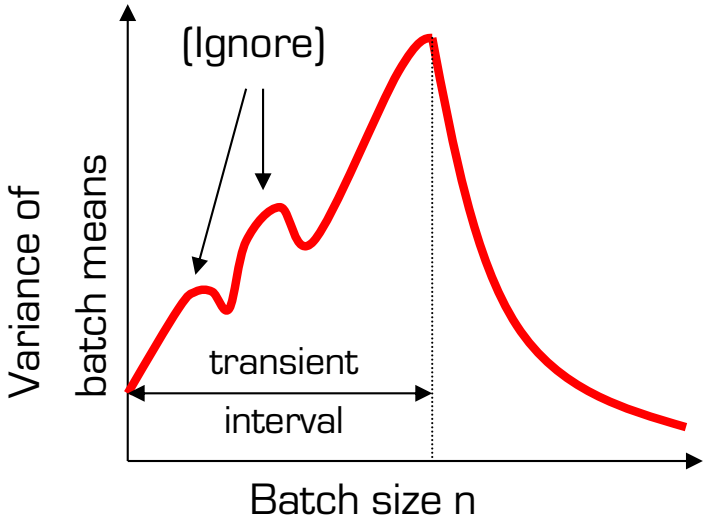
- Compute the variance of batch means as a function of batch size n

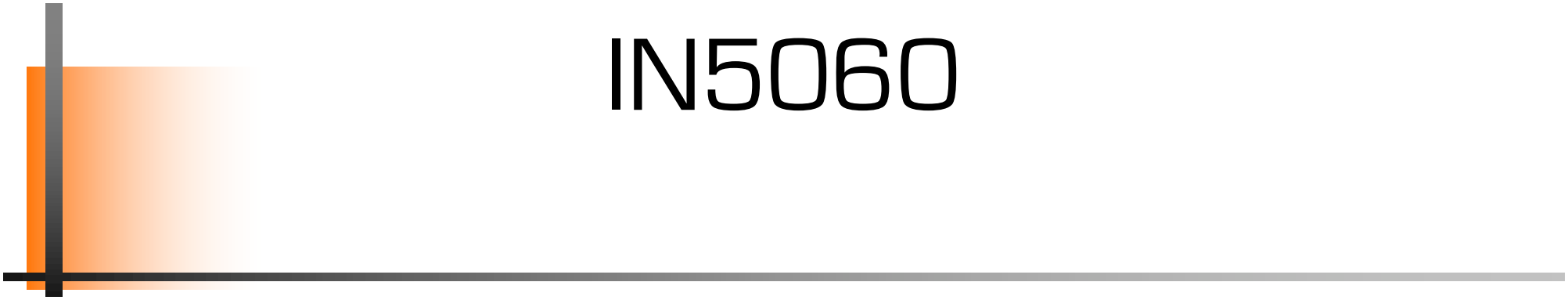
$$var(x) = \frac{1}{m-1}\sum_{i=1}^{m}(\bar{x}_i - \bar{\bar{x}})^2$$

- Plot variance versus size n

- When n starts decreasing, the end of the transient interval has been found







IN5060

Analysis of results
termination criteria

UiO : University of Oslo

Terminating Simulations

- For some simulations, transition state is of interest
- In these cases, transient removal cannot be performed

- Sometimes upon termination you also get final conditions that do not reflect steady state
  - Can apply transition removal conditions to end of simulation
  - very frequently in open-loop simulation where the remaining samples are “draining” from the system

Terminating Simulations

- Take care when gathering at end of simulation
  - For example mean service time should include only those process that finish
  - and not those that are either in a queue or being processed when the simulation ends

- Also, take care of values at event times
  - Ex: queue length needs to consider area under curve
  - Say $t=0$ two jobs arrive, $t=1$ first job leaves, $t=4$ second job leaves
  - queue lengths $q_0=2$, $q_1=1$, $q_4=0$ but q average not $(2+1+0)/3=1$
  - Instead, area is $2+1+1+1$ so q average $5/4=1.25$

Stopping Criteria

- Important to run long enough
  - Stopping too short may give variable results
  - Stopping too long may waste resources
- Should get confidence intervals on mean to desired width:

$$\bar{x} \pm z_{1-\frac{\alpha}{2}} var(\bar{x})$$

separate slide

- Variance of sample mean of independent observations:

$$var(\bar{x}) = \frac{1}{n} var(x)$$

- But only if observations independent! Most simulations not
  - Ex: if queuing delay for packet i is large then will likely be large for packet $i+1$
- So, use: independent replications, batch means, regeneration (all next)

Confidence interval

- The probability
- that the values drawn for an infinite series of samples
- stays within the boundaries around the computed average value
- with a confidence of $(1-\alpha)\cdot 100\%$
- is defined by the boundaries $\mathrm{x} \pm z_{1-\frac{\alpha}{2}} var(x)$
- where the values z_p are the quantities of the normal distribution
- values for can z_p be looked up in a table (Appendix A.2 in the Jain book)
  - for example for a 95-percentile
  - α =0.05
  - $p = 1 - \frac{\alpha}{2} = 0.975$
  - $z_{0.975} = 1.960$
- then compute the confidence interval as $1.96\ var(x)$
- or in Excel

  `=CONFIDENCE(0.05,sqrt(var(x)),1)`

  the first is alpha, the second the standard deviation [$\sqrt{var(x)}$], the unclear

Independent Replications

- Assume replications are independent
  - Different random seed values
- Collect m replications of size $n+n_0$ each
  - n_0 is length of transient phase
- Mean for each replication $\forall i: 1..m$

$$\bar{x}_i = \frac{1}{n} \sum_{j=n_0+1}^{n_0+n} x_{ij}$$

- Overall mean for all replications: $\bar{\bar{x}} = \frac{1}{m} \sum_{i=1}^{m} \bar{x}_i$
- Calculate variance of replicate means

$$var(\bar{x}) = \frac{1}{m-1} \sum_{i=1}^{m} (\bar{x}_i - \bar{\bar{x}})^2$$

- Confidence interval is

$$\bar{\bar{x}} \pm z_{1-\alpha/2} var(\bar{x})$$

- Note, width proportional to $\sqrt{mn}$, but reduce "waste" of mn_0 observations, increase length n

Batch Means

- Collect long run of N samples + n_0
  - n_0 is length of transient phase
- Divide into m batches of n observations each
  - n large enough so little correlation between
- Mean for each batch i $\quad \forall i: 1..m \quad$ $\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij}$
- Overall mean for all replications $\quad \bar{\bar{x}} = \frac{1}{m}\sum_{i=1}^{m} \bar{x}_i$
- Calculate variance of replicate means

$$var(\bar{x}) = \frac{1}{m-1}\sum_{i=1}^{m}(\bar{x}_i - \bar{\bar{x}})^2$$

- Confidence interval is

$$\bar{\bar{x}} \pm z_{1-\frac{\alpha}{2}} var(\bar{x})$$

- Note, similar to independent replications but less waste (only n_0)

Batch Means

- How to choose n? Want covariance of successive means small compared to variance

$$cov(\overline{x_i}, \overline{x_{i+1}}) = \frac{1}{m-2}\sum_{i=1}^{m}\left((\overline{x_i} - \bar{x})(\overline{x_{i+1}} - \bar{x})\right)$$

- Start $n=1$, then double n

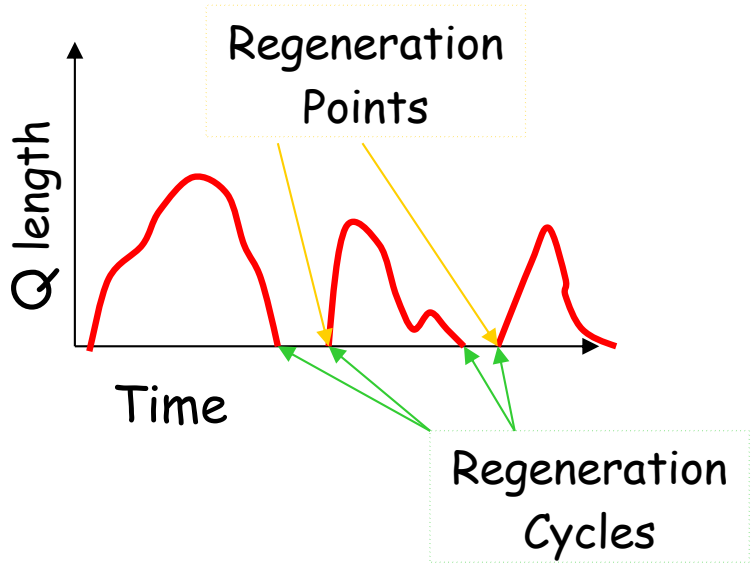
- Example:

Size	Cov	Var
1	-0.187	1.799
2	0.026	0.811
4	0.110	0.420
...		
64	0.00010	0.06066

- Becomes less than 1%, so $n=64$ brings us beyond the distance where x and $x+n$ are dependent
- so can use $n=64$ as batch size

Method of Regeneration

- Consider CPU scheduling
- Jobs arriving after queue empty not dependent upon previous
- Note, system with two queues would need both to be idle
- Not all systems are regenerative
  - Not those with “long” memories
- Note, unlike in batch methods, the cycles can be of different lengths



Method of Regeneration

- m cycles of length $n_1, n_2, \ldots, n_m$
- jth observation in cycle i: x_{ij}
- cycle means $\bar{x}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} x_{ij}$
- Note, for the overall mean, we known that

$$\bar{\bar{x}} \neq \frac{1}{m}\sum \bar{x}_i$$

- since cycles have different lengths. So, to compute confidence intervals
  - Compute sums:

$$y_i = \sum_{j=1}^{n_i} x_{ij}$$

  - Compute overall mean:

$$\bar{\bar{x}} = \frac{\sum_{i=1}^{m} y_i}{\sum_{i=1}^{m} n_i}$$

  - Calculate difference between mean and observed sums:

$$w_i = y_i - n_i\bar{\bar{x}} \qquad \forall i \in \{1..m\}$$

  - Calculate variance of differences:

$$var(w) = \frac{1}{m-1}\sum_{i=1}^{m} w_i^2$$

  - Compute mean cycle length:

$$\bar{n} = \frac{1}{m}\sum_{i=1}^{m} n_i$$

  - Confidence interval:

$$\bar{\bar{x}} \pm z_{1-\alpha/2}\frac{var(w)}{\bar{n}\sqrt{m}}$$

Question

- Imagine you are called in as an expert to review a simulation study. Which of the following would you consider non-intuitive and would want extra validation?
  1. Throughput increases as load increases
  2. Throughput decreases as load increases
  3. Response time increases as load increases
  4. Response time decreases as load increases
  5. Loss rate decreases as load increases