Scheduling
CS 111
Operating Systems
Peter Reiher

Outline

- What is scheduling?
  - What are our scheduling goals?
- What resources should we schedule?
- Example scheduling algorithms and their implications

What Is Scheduling?

- An operating system often has choices about what to do next
- In particular:
  - For a resource that can serve one client at a time
  - When there are multiple potential clients
  - Who gets to use the resource next?
  - And for how long?
- Making those decisions is scheduling

OS Scheduling Examples

- What job to run next on an idle core?
  - How long should we let it run?
- In what order to handle a set of block requests for a disk drive?
- If multiple messages are to be sent over the network, in what order should they be sent?

How Do We Decide How To Schedule?

- Generally, we choose goals we wish to achieve
- And design a scheduling algorithm that is likely to achieve those goals
- Different scheduling algorithms try to optimize different quantities
- So changing our scheduling algorithm can drastically change system behavior

The Process Queue

- The OS typically keeps a queue of processes that are ready to run
  - Ordered by whichever one should run next
  - Which depends on the scheduling algorithm used
- When time comes to schedule a new process, grab the first one on the process queue
- Processes that are not ready to run either:
  - Aren't in that queue
  - Or are at the end
  - Or are ignored by scheduler

Potential Scheduling Goals

- Maximize throughput
  - Get as much work done as possible
- Minimize average waiting time
  - Try to avoid delaying too many for too long
- Ensure some degree of fairness
  - E.g., minimize worst case waiting time
- Meet explicit priority goals
  - Scheduled items tagged with a relative priority
- Real time scheduling
  - Scheduled items tagged with a deadline to be met

Different Kinds of Systems, Different Scheduling Goals

- Time sharing
  - Fast response time to interactive programs
  - Each user gets an equal share of the CPU
- Batch
  - Maximize total system throughput
  - Delays of individual processes are unimportant
- Real-time
  - Critical operations must happen on time
  - Non-critical operations may not happen at all

Preemptive Vs. Non-Preemptive Scheduling

- When we schedule a piece of work, we could let it use the resource until it finishes
- Could use virtualization to interrupt part way through
  - Allowing other pieces of work to run instead
- If scheduled work always runs to completion, the scheduler is non-preemptive
- If the scheduler temporarily halts running jobs to run something else, it's preemptive
- Cooperative scheduling – when process blocks or voluntarily releases, schedule someone else

Pros and Cons of Non-Preemptive Scheduling

+ Low scheduling overhead
+ Tends to produce high throughput
+ Conceptually very simple
– Poor response time for processes
– Bugs can cause machine to freeze up
  – If process contains infinite loop, e.g.
– Not good fairness (by most definitions)
– May make real time and priority scheduling difficult

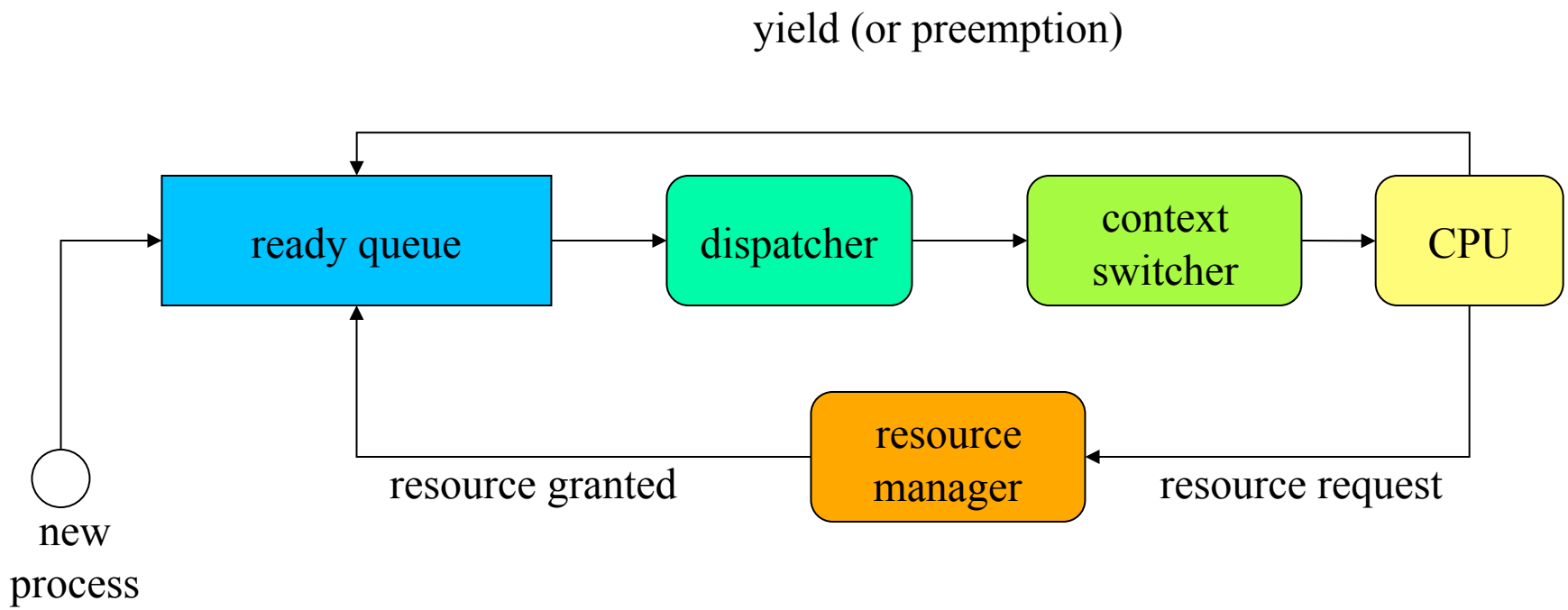
Pros and Cons of Pre-emptive Scheduling

+ Can give good response time
+ Can produce very fair usage
+ Works well with real-time and priority scheduling
– More complex
– Requires ability to cleanly halt process and save its state
– May not get good throughput

Scheduling: Policy and Mechanism

- The scheduler will move jobs into and out of a processor (*dispatching*)
  - Requiring various mechanics to do so
- How dispatching is done should not depend on the policy used to decide who to dispatch
- Desirable to separate the choice of who runs (policy) from the dispatching mechanism
  - Also desirable that OS process queue structure not be policy-dependent

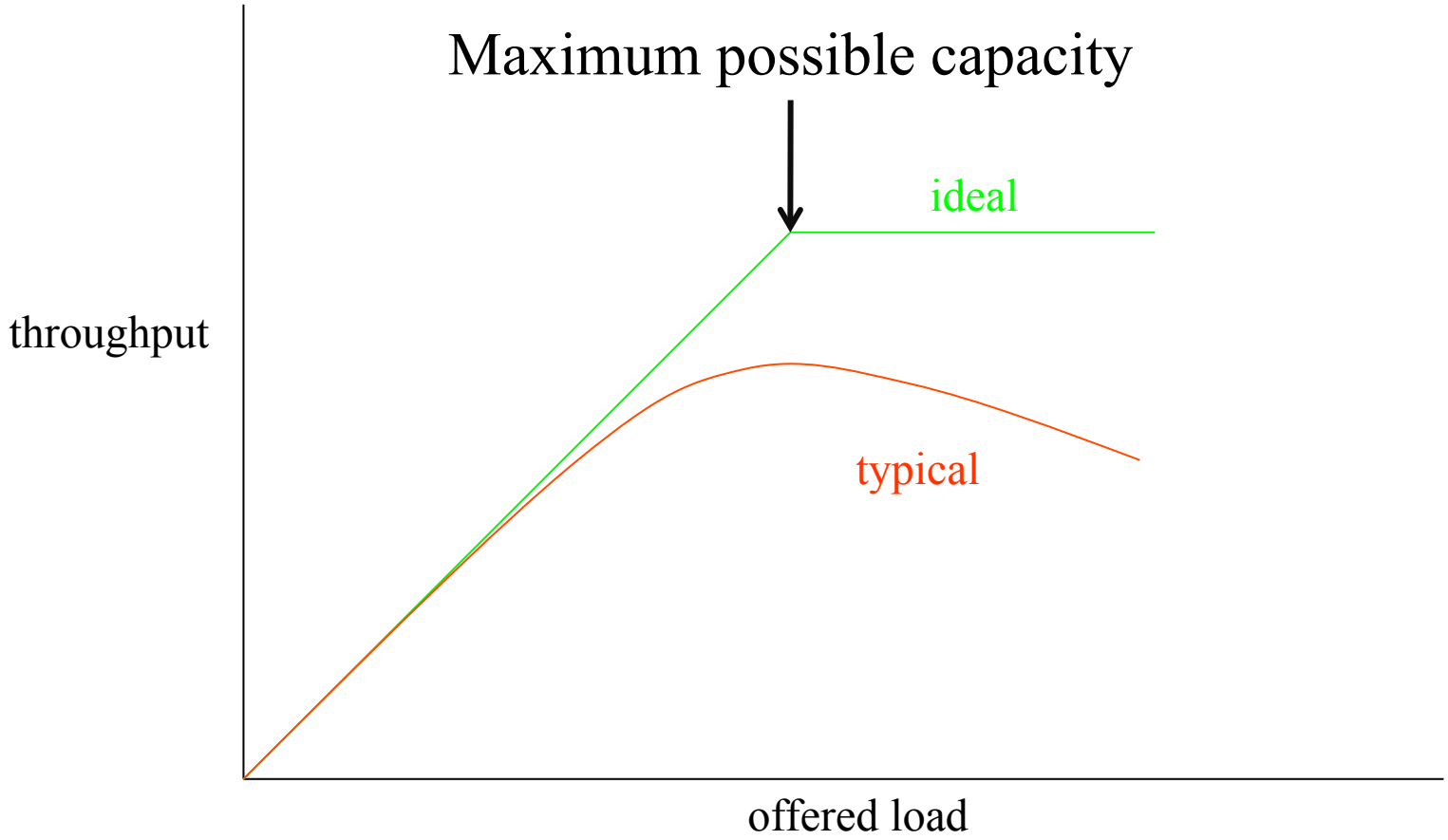
Scheduling the CPU



Scheduling and Performance

- How you schedule important system activities has a major effect on performance
- Performance has different aspects
  - You may not be able to optimize for both
- Scheduling performance has very different characteristic under light vs. heavy load
- Important to understand the performance basics regarding scheduling

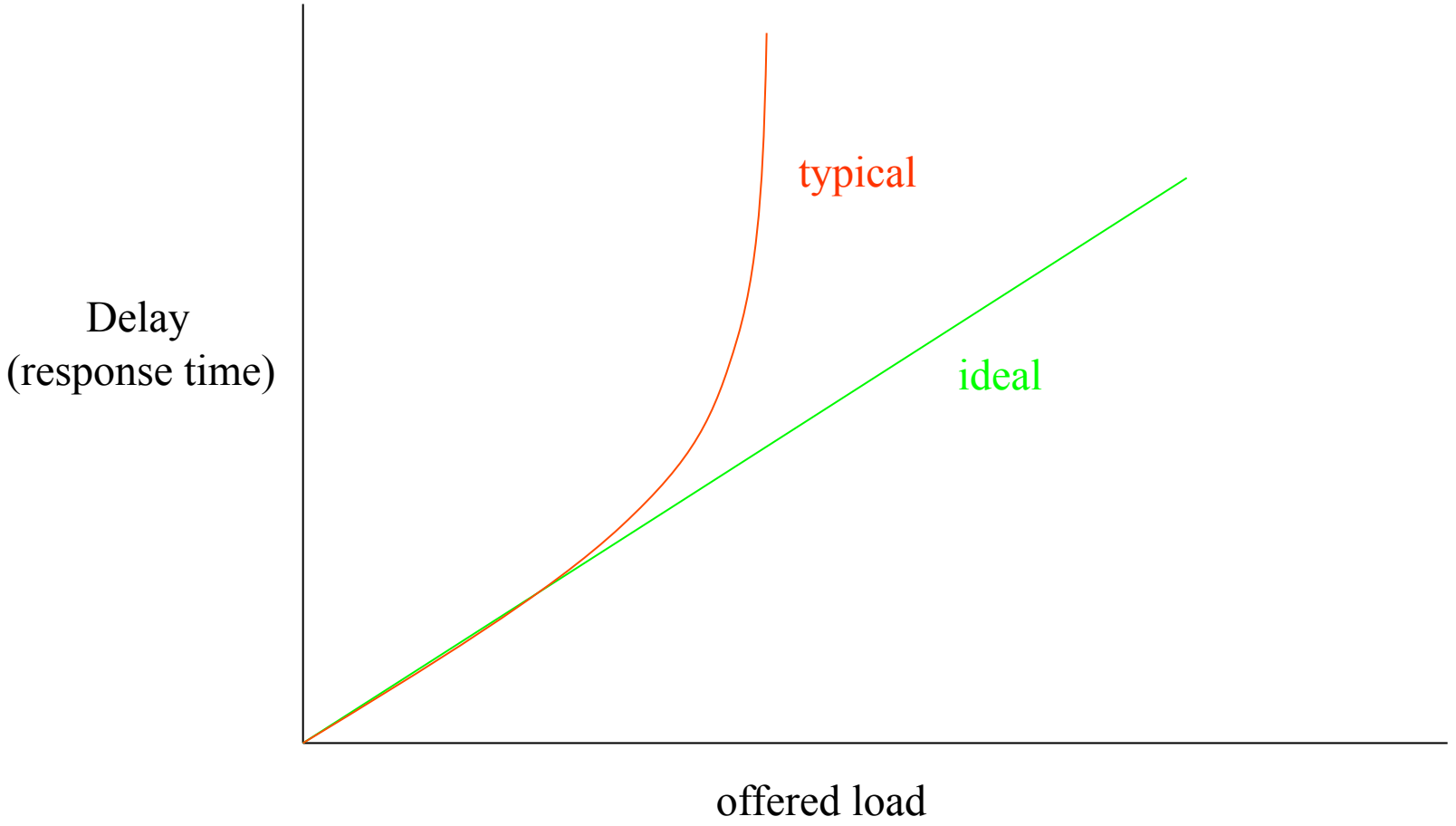
Typical Throughput vs. Load Curve



Why Don't We Achieve Ideal Throughput?

- Scheduling is not free
  - It takes time to dispatch a process (overhead)
  - More dispatches means more overhead (lost time)
  - Less time (per second) is available to run processes
- How to minimize the performance gap
  - Reduce the overhead per dispatch
  - Minimize the number of dispatches (per second)
- This phenomenon is seen in many areas besides process scheduling

Typical Response Time vs. Load Curve



Why Does Response Time Explode?

- Real systems have finite limits
  - Such as queue size
- When those limits are exceeded, requests are typically dropped
  - Which is an infinite response time, for them
  - There may be automatic retries (e.g., TCP), but they could be dropped, too
- If load arrives a lot faster than it is serviced, lots of stuff gets dropped
- Unless careful, overheads during heavy load explode
- Effects like receive livelock can also hurt

Graceful Degradation

- When is a system "overloaded"?
  - When it is no longer able to meet service goals
- What can we do when overloaded?
  - Continue service, but with degraded performance
  - Maintain performance by rejecting work
  - Resume normal service when load drops to normal
- What should we not do when overloaded?
  - Allow throughput to drop to zero (i.e., stop doing work)
  - Allow response time to grow without limit