CSCI 690 – Week 5

Toward Ops: Performance, Reliability, Monitoring

Acknowledgements

- Jez Humble
- Armando Fox and David Patterson

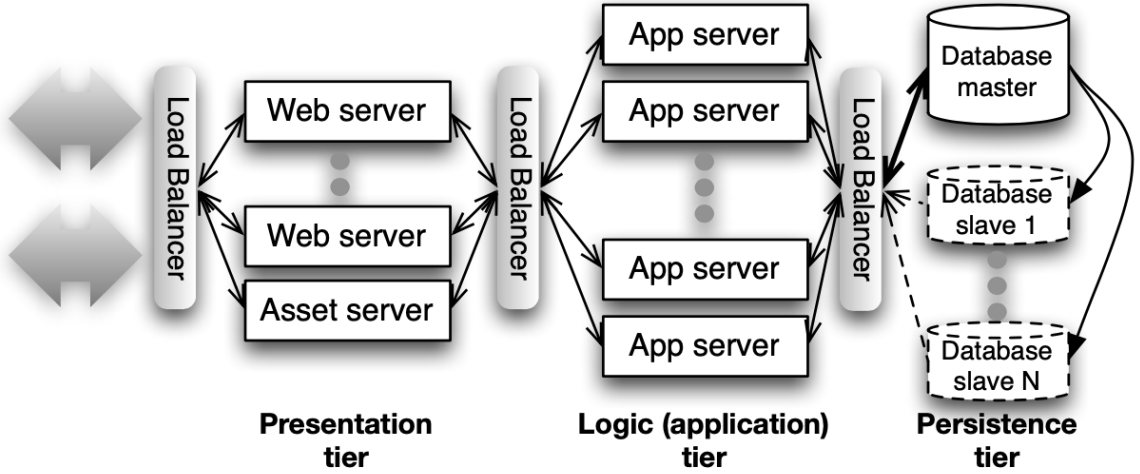
Testing non-functional requirements

Capacity, throughput, performance

NFRs are a significant delivery risk to be mitigated

Three Tier Architecture

- **Tiers**:
  - Presentation: Render views + interact with users
  - Logic/Application: Runs SaaS app code + app logic
  - Persistence: Stores app data
- **"Shared Nothing"**: Can add more computers to presentation + logic tiers for scale.



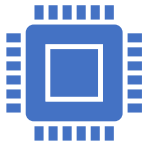
“Performance & security” defined

- Availability or Uptime
  What % of time is site up & accessible?
- Responsiveness
  - How long after a click does user get response?
- Scalability
  - As # users increases, can you maintain responsiveness without increasing cost/user?
- Privacy
  - Is data access limited to the appropriate users?
- Authentication
  - Can we trust that user is who s/he claims to be?
- Data integrity
  - Is users’ sensitive data tamper-evident?

Performance
Stability

Security

Measuring capacity



Scalability testing

Change in response time as we add more servers, services, or threads

Longevity testing

Long-time run to catch memory leaks or stability problems

Throughput testing

How many transactions per second can the system handle

Load testing

What happens to capacity when the load increases significantly

Automating capacity testing in pipeline

- Most capacity tests aren’t for the commit stages of pipeline
- Some can be run in parallel with acceptance tests
- Usually best *after* the acceptance phase
- Most effective in isolated environment

Quantifying Availability and Responsiveness

(a little more on the Ops side of DevOps)

Monitoring strategy

- Instrument apps and infrastructure to collect needed data
- Store the data so it is easy to analyze
- Aggregate data with dashboards
- Use notifications/alerts

Availability and Response time

- Gold standard: US public phone system, 99.999% uptime (“five nines”)
  - Rule of thumb: 5 nines ~ 5 minutes/year
  - Since each nine is an order of magnitude, 4 nines ~ 50 minutes/year, etc.
  - Good Internet services get 3-4 nines
- Response time: how long after I interact with site do I perceive response?
  - For small content on fast network, dominated by latency (not bandwidth)

Is response time important?

- How important is response time?*
  - Amazon: +100ms => 1% drop in sales
  - Yahoo!: +400ms => 5-9% drop in traffic
  - Google: +500ms => 20% fewer searches
- Classic studies (Miller 1968, Bhatti 2000)

  <100 ms is "instantaneous"

  >7 sec is abandonment time

Jeff Dean,
Google Fellow

"Speed is a feature"

Source: Nicole Sullivan (Yahoo! Inc.), *Design Fast Websites*, http://www.slideshare.net/stubbornella/designing-fast-websites-presentation

Service Level Objective (SLO)

- Time to satisfy user request
  ("latency" or "response time")
- SLO: Instead of worst case or average: what % of users get acceptable performance
- Specify %ile, target response time, time window
  - e.g., 99% < 1 sec, over a 5-minute window
  - why is time window important?
- Service level *agreement* (SLA) is an SLO to which provider is contractually obligated

Apdex: simplified SLO

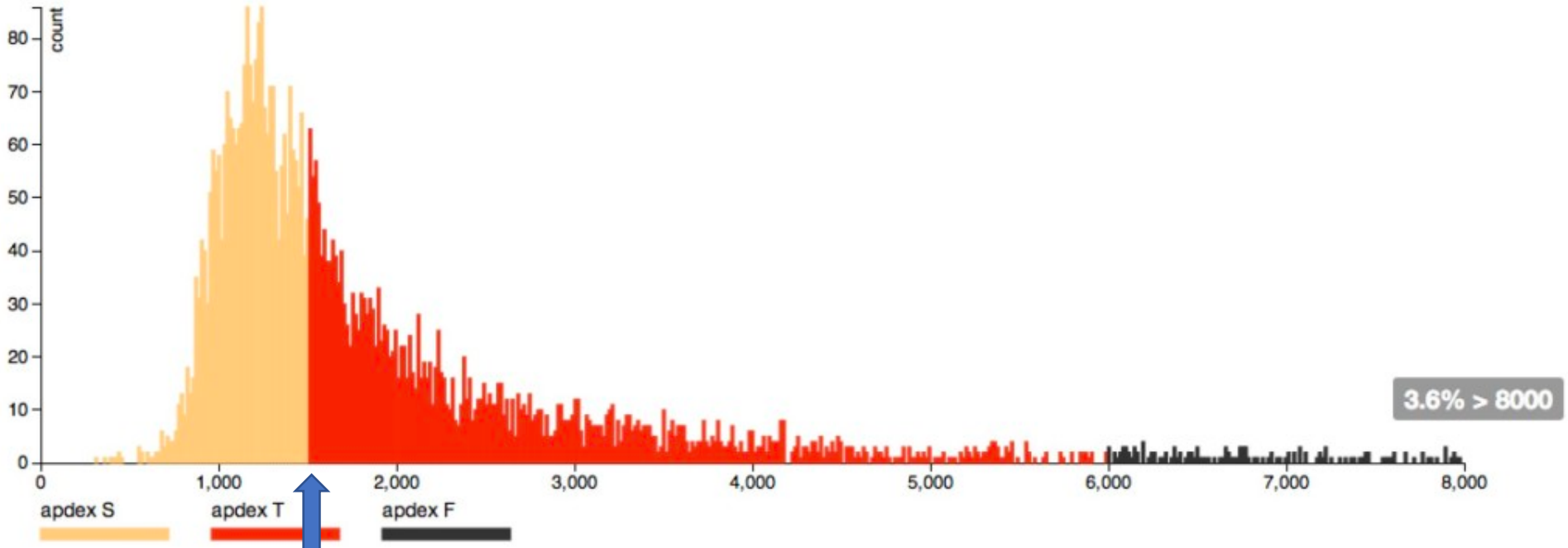
Given a threshold latency *T* for user satisfaction:

Satisfactory requests take $t \le T$	*Tolerable* requests take $T \le t \le 4T$	Apdex = (#sat + 0.5(#tol)) / #reqs	0.85 to 0.93 generally "good"

Warning! Can hide *systematic* outliers if not used carefully!

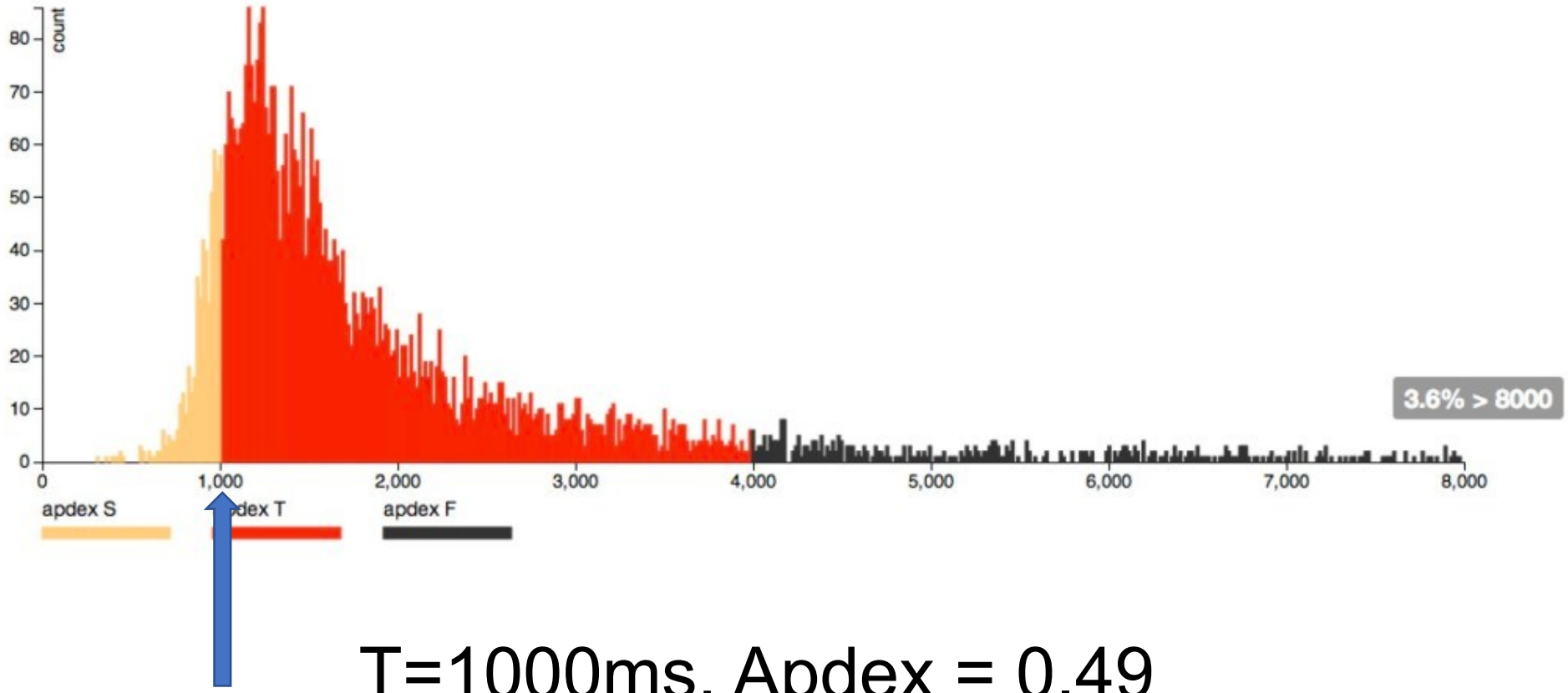
e.g. critical action occurs once in every 15 clicks but takes 10x as long => (14+0)/15 > 0.9

Apdex Visualization



T=1500ms, Apdex = 0.7

Apdex Visualization



T=1000ms, Apdex = 0.49

Monitoring

(ESaaS §12.5)

Armando Fox

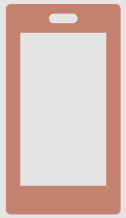
Kinds of monitoring

- "If you're not monitoring it, it's probably broken"
- At development time (*profiling*)
  - Identify possible performance/stability problems *before* they get to production
- In production
  - Internal: instrumentation embedded in app and/or framework (Rails, Rack, etc.)
  - External: active probing by other site(s).

Internal monitoring

pre-SaaS/PaaS: *local*

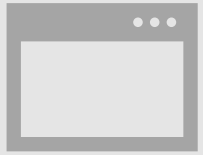
Info collected & stored locally, e.g., Nagios



Today: *hosted*

Info collected in your app but stored centrally

Info available even when app is down



Example: New Relic

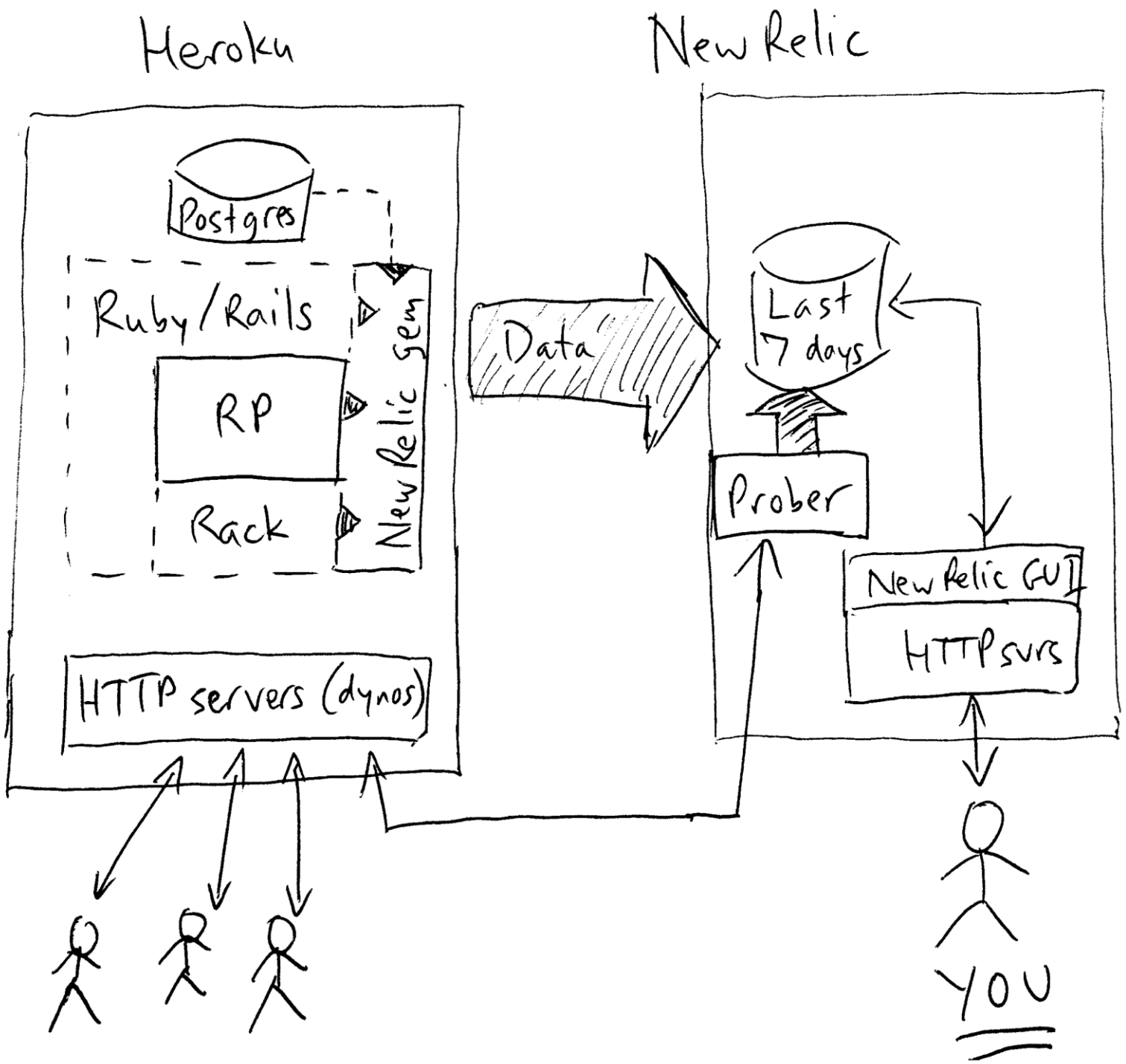
conveniently, has both a development mode and production mode

basic level of service is? (was) free for Heroku apps

Why use external monitoring?

- Detect if site is down
- Detect if site is slow for reasons outside measurement boundary of internal monitoring
- Get user's view from many different places on the Internet
- Example: Pingdom

New Relic example



Plan-and-Document + Reliability

- Dependability via redundancy
  - Guideline: no single point of failure
- How much redundancy can customer afford?
- *Mean Time To Failure (MTTF)* includes SW & operators as well as HW
- Unavailability ≈ *Mean Time To Repair*/MTTF
  - Improving MTTR may be easier than MTTF, but can try to improve both MTTR & MTTF

P&D + Processes to Improve SW

- P&D assumption is can improve SW development process of organization
  => More reliable SW product
  - Record all aspects of project
    to see what can improve
- Get ISO 9001 standard if a company has
  1. Process in place
  2. Method to see if process is followed
  3. Records results to improve process
  - Approval for *process*, not quality of resulting code

Managing continuous delivery

More than tools and automation work

Have a clear view of the objectives of organizational change

Maturity model for configuration and release management (Humble + Farley)

Practice	Build management and continuous integration	Environments and deployment	Release management and compliance	Testing	Data management
Level 3 - Optimizing: Focus on process improvement	Teams regularly meet to discuss integration problems and resolve them with automation, faster feedback, and better visibility.	All environments managed effectively. Provisioning fully automated. Virtualization used if applicable.	Operations and delivery teams regularly collaborate to manage risks and reduce cycle time.	Production rollbacks rare. Defects found and fixed immediately.	Release to release feedback loop of database performance and deployment process.
Level 2 - Quantitatively managed: Process measured and controlled	Build metrics gathered, made visible, and acted on. Builds are not left broken.	Orchestrated deployments managed. Release and rollback processes tested.	Environment and application health monitored and proactively managed. Cycle time monitored.	Quality metrics and trends tracked. Non functional requirements defined and measured.	Database upgrades and rollbacks tested with every deployment. Database performance monitored and optimized.
Level 1 - Consistent: Automated processes applied across whole application lifecycle	Automated build and test cycle every time a change is committed. Dependencies managed. Re-use of scripts and tools.	Fully automated, self-service push-button process for deploying software. Same process to deploy to every environment.	Change management and approvals processes defined and enforced. Regulatory and compliance conditions met.	Automated unit and acceptance tests, the latter written with testers. Testing part of development process.	Database changes performed automatically as part of deployment process.
Level 0 – Repeatable: Process documented and partly automated	Regular automated build and testing. Any build can be re-created from source control using automated process.	Automated deployment to some environments. Creation of new environments is cheap. All configuration externalized / versioned	Painful and infrequent, but reliable, releases. Limited traceability from requirements to release.	Automated tests written as part of story development.	Changes to databases done with automated scripts versioned with application.
Level -1 – Regressive: processes unrepeatable, poorly controlled, and reactive	Manual processes for building software. No management of artifacts and reports.	Manual process for deploying software. Environment-specific binaries. Environments provisioned manually.	Infrequent and unreliable releases.	Manual testing after development.	Data migrations unversioned and performed manually.

Success needs good management

Create and improve processes for efficient software delivery

Manage and mitigate risks

Diagnose and fix issues

Toward Ops?

This week

- Finish module 4 videos, quiz, and HW (Friday)
- Discussion forum: Site Reliability Engineers vs. DevOps Engineers
- New HW3
- Project details posted by Friday

Next week

- June 14th VIRTUAL guest lecture (Zoom), 530pm
  - DevSecOps
  - General Q+A (please prepare)
  - Watch for possible pre-class reading
- Take-home final posted Wednesday 15th (due Wednesday June 22nd)
- No discussion posts (work on exam/projects instead)

Final week

- Project demos/presentations Tuesday 21st
- Final project submission NLT 1159pm Wednesday 22nd
- Final exam submission NLT 1159pm Wednesday 22nd
- Final grades posted NLT Tuesday 28th 10am (hopefully earlier)