

**Fast by default:
to achieve sustainable high performance**

**Xiaoliang “David” Wei
Facebook Inc.
www.facebook.com/DavidWei
DavidWei@acm.org**

Velocity China, Dec 8th, 2010, Beijing

Agenda

1 Facebook: always moving fast

2 Focus on abstracts

3 Data-driven

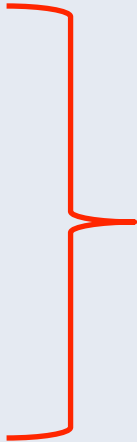
4 Empower the engineering team

Facebook: always moving fast

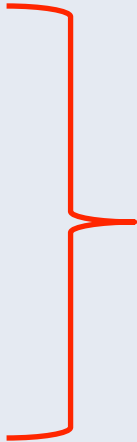
Facebook: fast evolution

- User adoption is always evolving
- One new revision each week
- Patches pushed everyday
- Urgent fixes 24/7

Facebook: fast evolution

- User adoption is always evolving
 - One new revision each week
 - Patches pushed everyday
 - Urgent fixes 24/7
- 
- Product cycle: in weeks

Facebook: fast evolution

- User adoption is always evolving
 - One new revision each week
 - Patches pushed everyday
 - Urgent fixes 24/7
- 
- Product cycle: in weeks

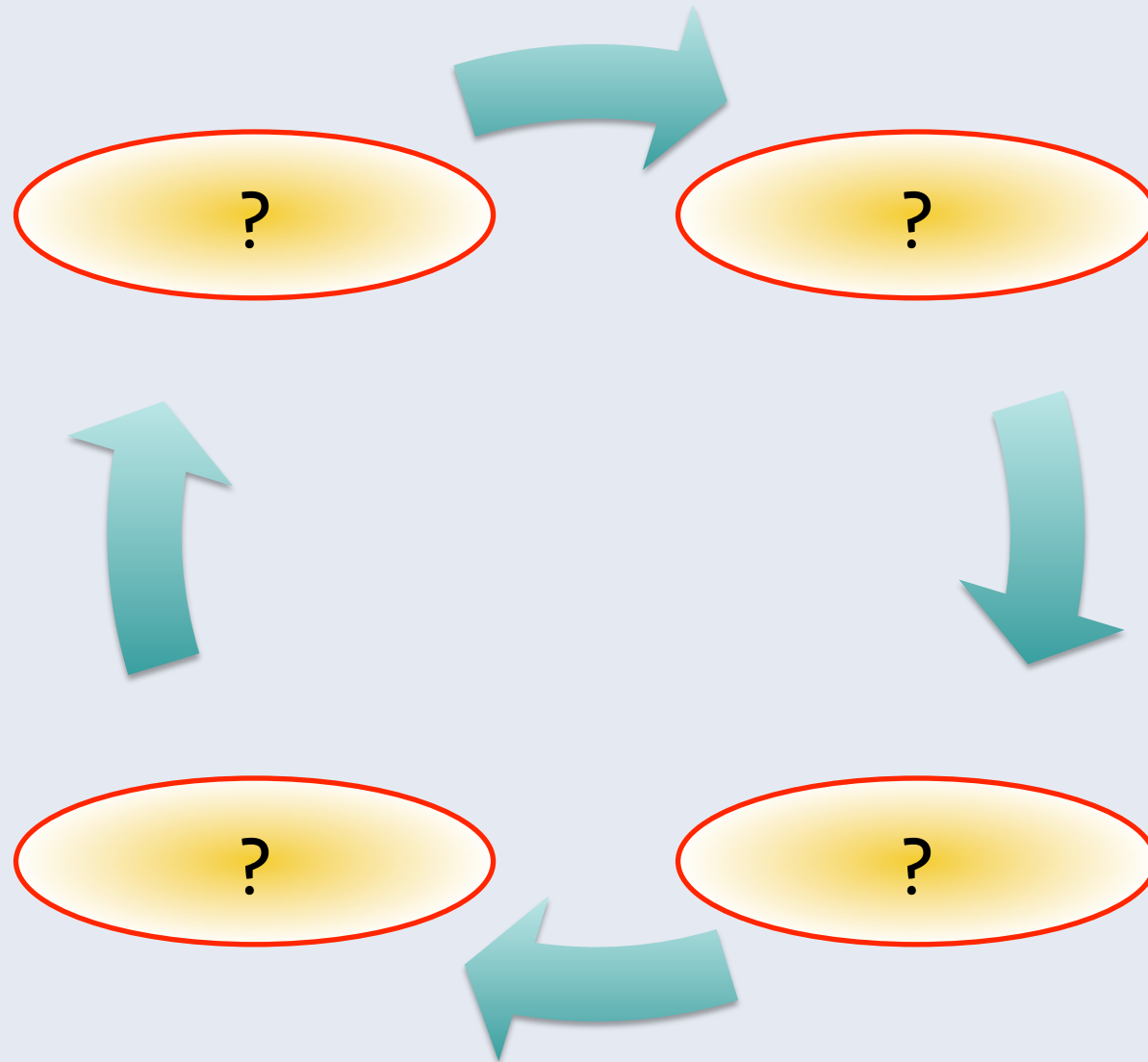
ALWAYS MOVING FAST

“Fast by default”

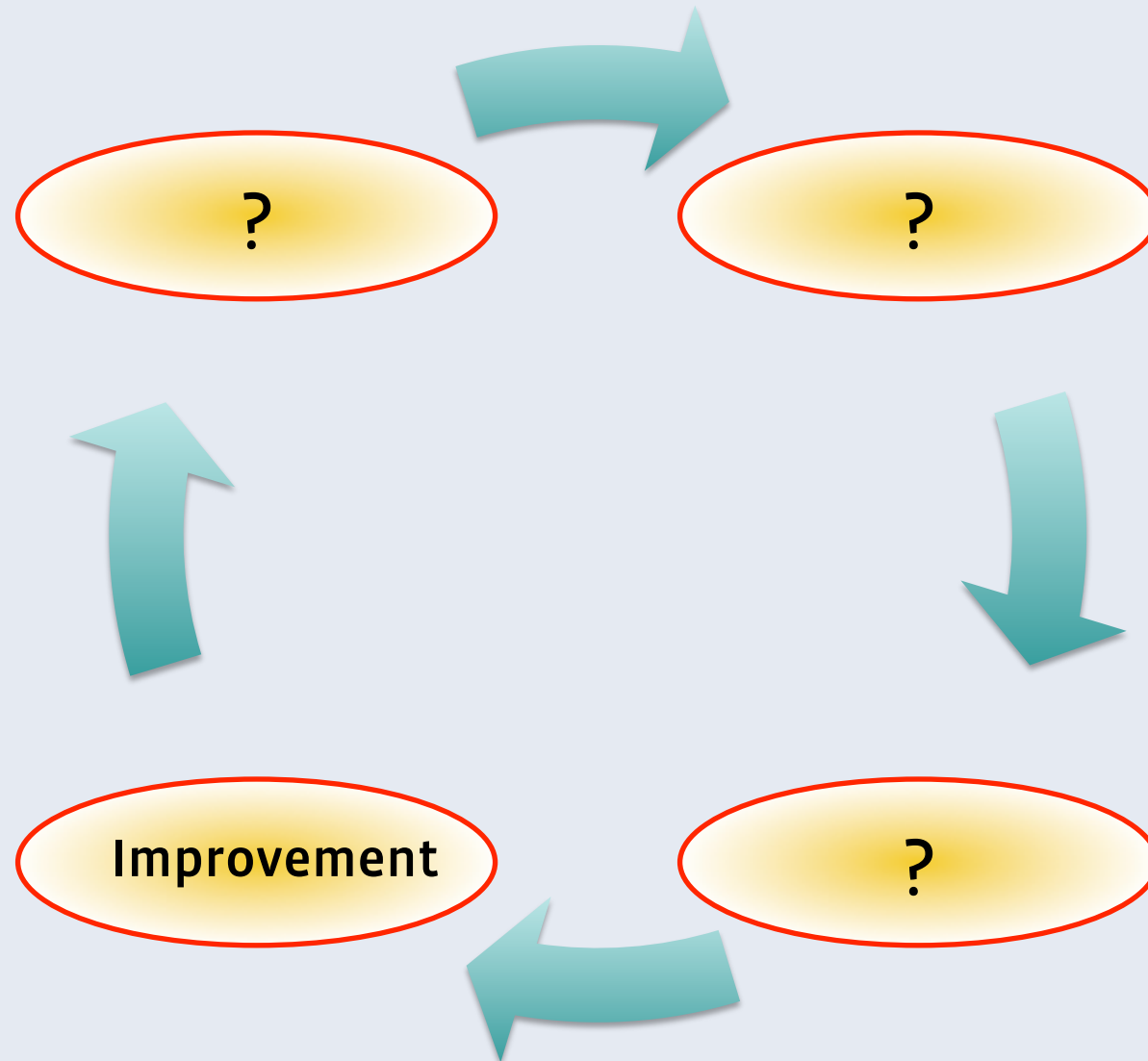
Making Facebook sustainably fast

- Mar 2008 – Jul 2008: Perf SWAT 20+ people
 - Aug 2008 – Jun 2009: focus on infrastructure 3~4 people
 - Jul 2009 – Dec 2009: Perf as a company goal ~ 10 people
 - 2010: Empowering the whole engineering team ~ 10 people
-
- How to make Facebook sustainably fast by a small team on performance and frontend infrastructure?

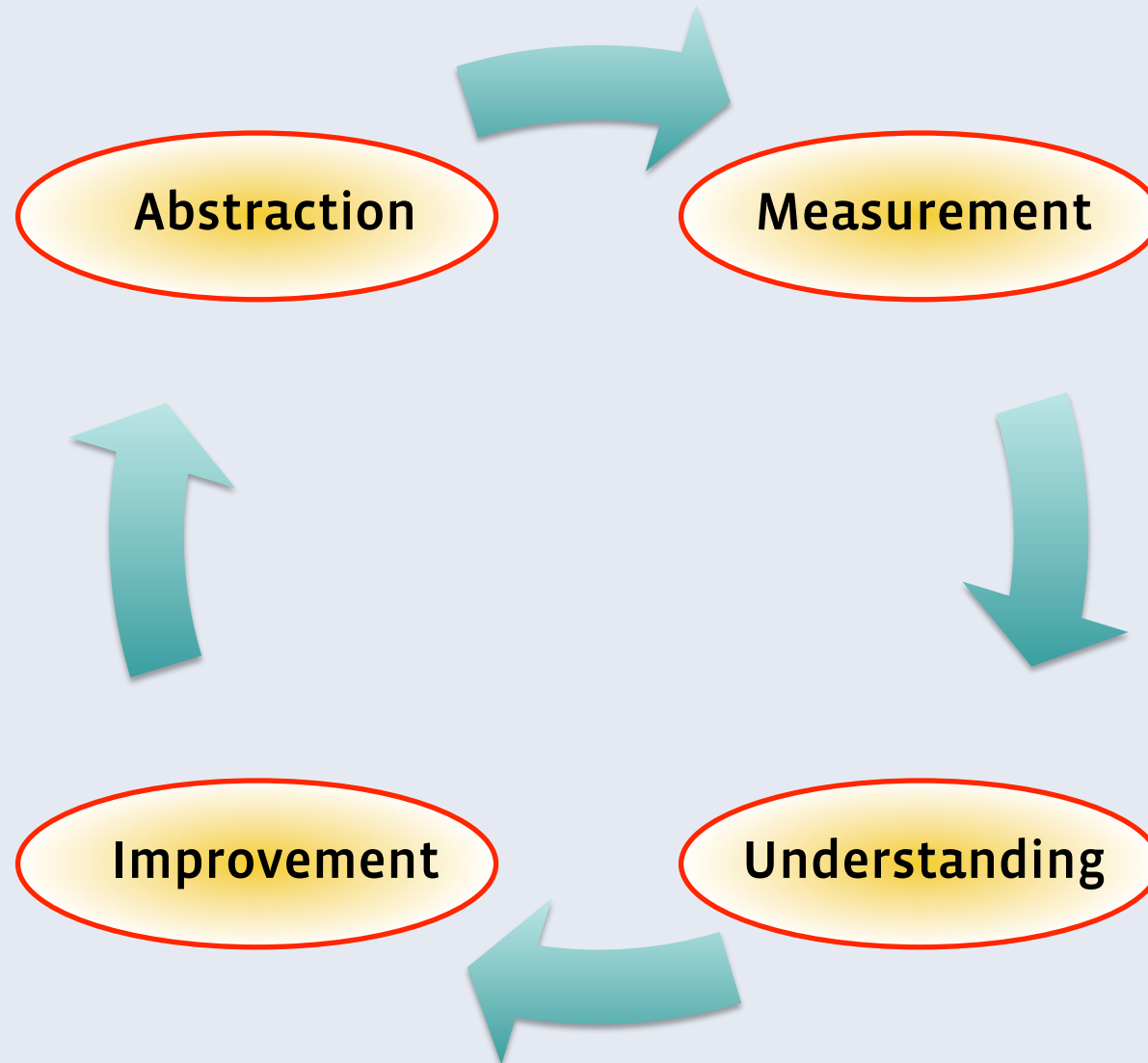
The cycle of web performance improvement



The cycle of web performance improvement



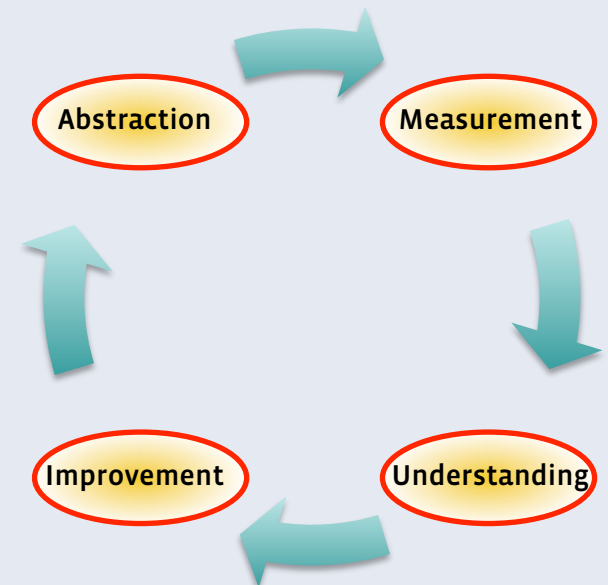
The cycle of web performance improvement



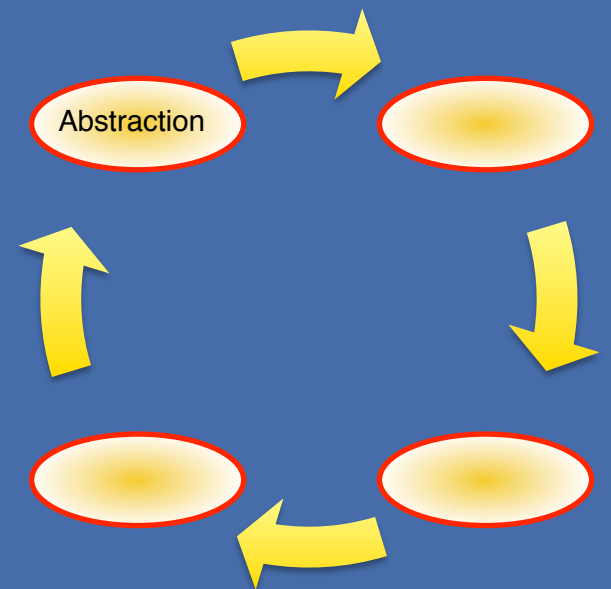
“Fast by default”

Making Facebook sustainably fast

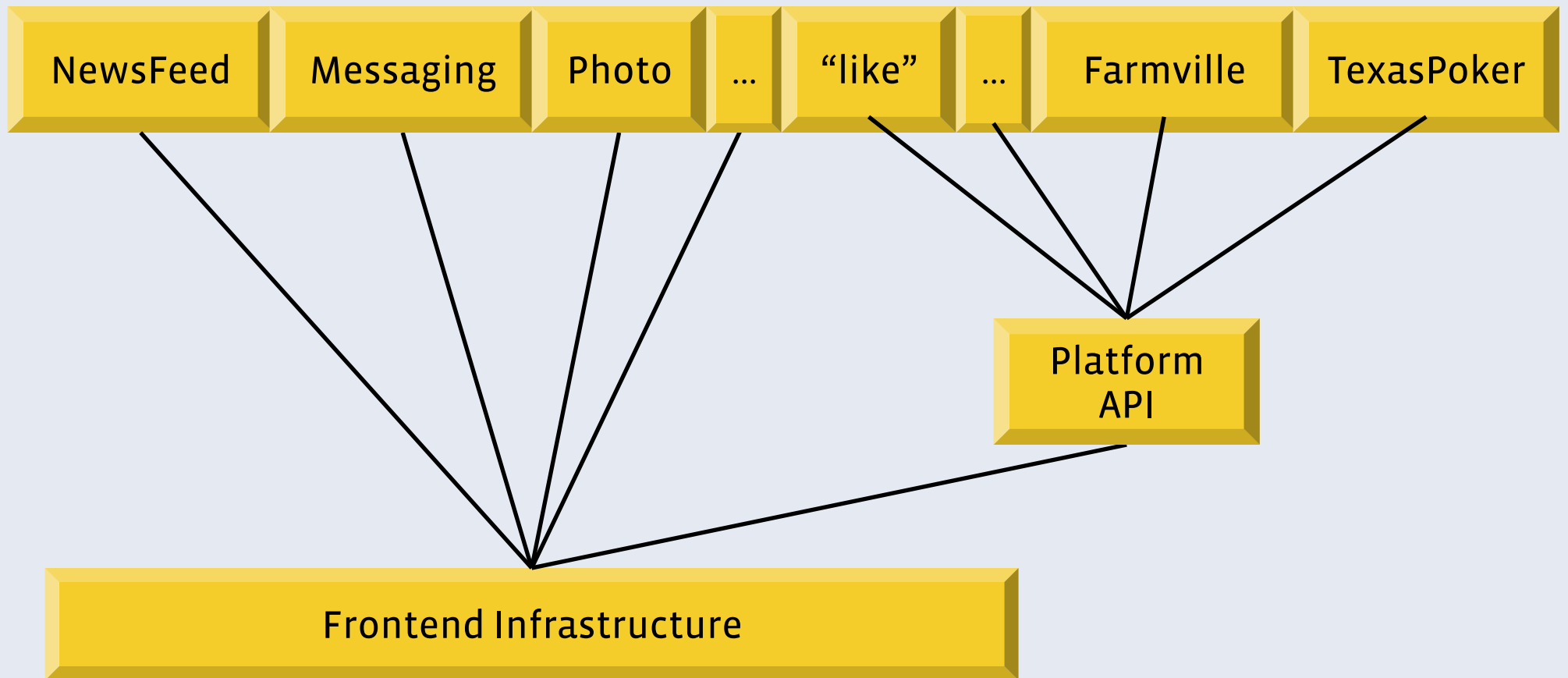
- Focus on abstracts
- Data driven
- Empowering the whole engineering team



Focus on abstracts



Abstraction



Abstraction

Design principle

- Allow developer focus on product and move fast
 - Only one way to do one thing
 - A clear set of best-practice rules to follow
 - Hide details of performance and reliability tunings

Abstraction

Design principle

- Allow developer focus on product and move fast
 - Only one way to do one thing
 - A clear set of best-practice rules to follow
 - Hide details of performance and reliability tunings

Dream

- Product development should be as easy & fun as building *LEGO*

Abstraction

Example 1: Static Resource Management

- Only one way to do one thing:
 - At least 4 ways to load Javascripts --> “require_static”
- Clear set of best-practice rules:
 - “no inline JS script tag”
 - “no manual packaging”
- Hide details of performance and reliability tunings:
 - System optimizes the delivery of static resources (automatically choosing dynamic script tag injection and etc)

Abstraction

Example 2: User tracking

- Only one way to do one thing:
 - HTTP Cookie / Cookie / Server-side storage => Server-side cookie
- Clear set of best-practice rules:
 - “no product-specific cookies”
 - “Yummy Yummy... Your cookie is eaten by the Cookie Monster.”
- Hide details of performance and reliability tunings:
 - Server-side cookie takes care of data storage/propagation and user tracking

Abstraction

Example 3: Javascript Primer

- Only one way to do one thing:
 - href + onclick / event delegation => primer
- Clear set of best-practice rules:
 - “no inline javascript for event handler”
- Hide details of performance and reliability tunings:
 - System optimizes the JS / non-JS experience
 - System can optimize the pre-fetching / packaging of necessary JS for interactions

Primer: dialog links

Example:

```
<a rel="dialog" href="/ajax/intl/language_dialog.php">English (US)</a>
```



In the endpoint:

```
$dialog =  
    new DialogResponse(kAsync_Auth_Any, false  
    ...  
$dialog->setTitle($title)  
    ->setBody($body)  
    ->setButtons(DialogResponse::CLOSE)  
    ->send();
```



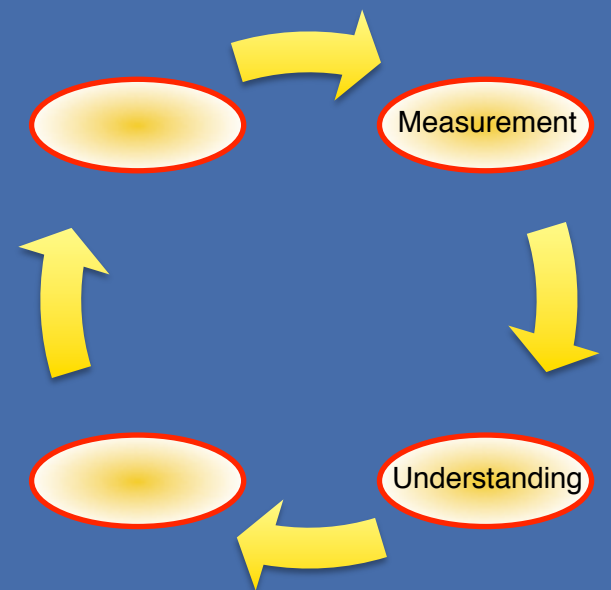
(by Makinde Adeagbo)

Abstraction

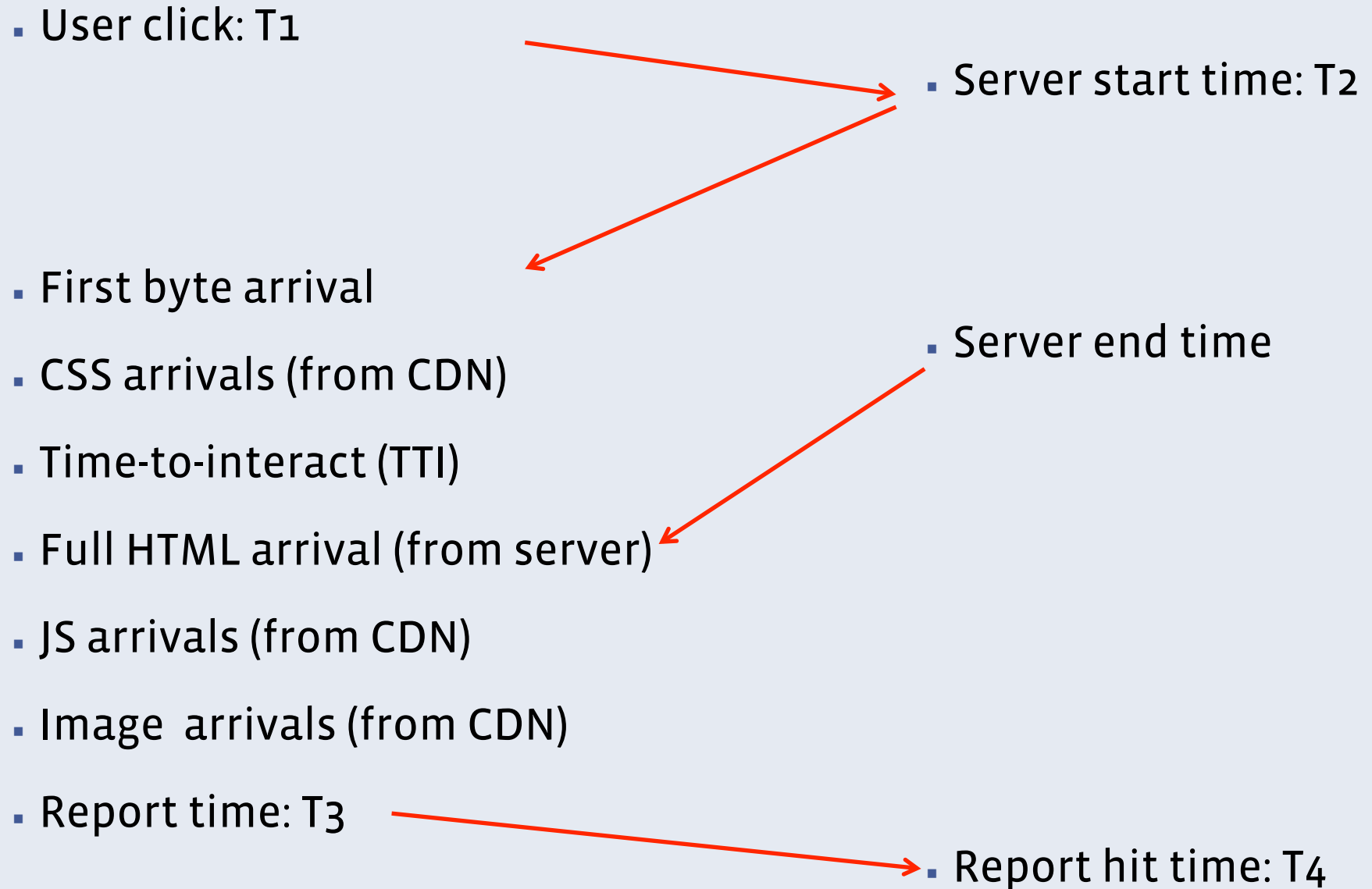
Examples:

- Static Resource Management
- User tracking: Server-side Cookie
- Interaction: Javascript Primer
- Page compositions: Pagelets & XHP
- PHP Preparable

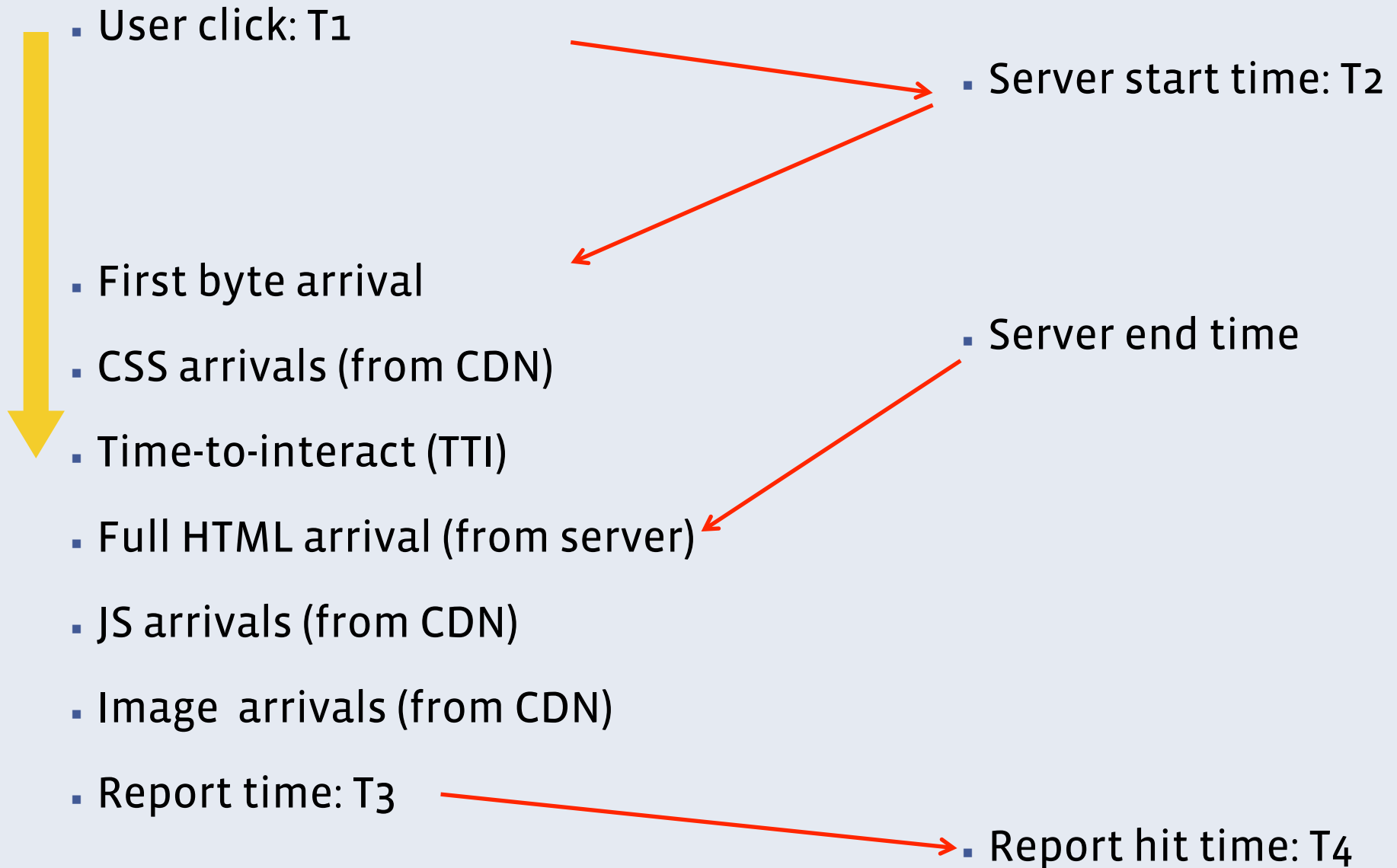
Data driven



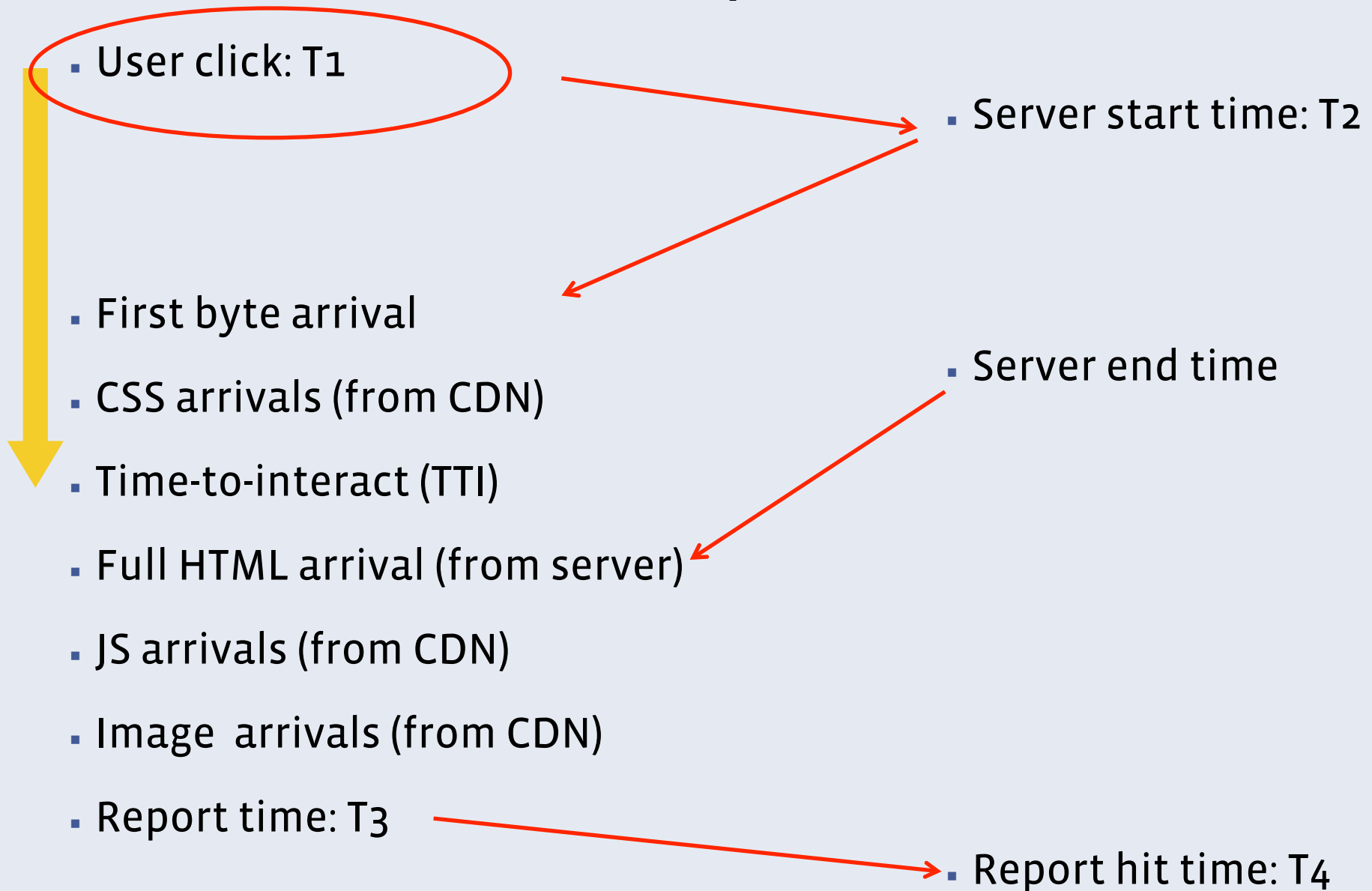
Measurement: End-to-end performance



Measurement: End-to-end performance



Measurement: End-to-end performance



Measurement: End-to-end performance

▪ User click: T₁

▪ Server start time: T₂

▪ First byte arrival

▪ CSS arrivals (from CDN)

▪ Time-to-interact (TTI)

▪ Full HTML arrival (from server)

▪ JS arrivals (from CDN)

▪ Image arrivals (from CDN)

▪ Report time: T₃

▪ Server end time

▪ Report hit time: T₄

Measurement: End-to-end performance

▪ User click: T₁

▪ Server start time: T₂

▪ First byte arrival

▪ CSS arrivals (from CDN)

▪ Time-to-interact (TTI)

▪ Full HTML arrival (from server)

▪ JS arrivals (from CDN)

▪ Image arrivals (from CDN)

▪ Report time: T₃

▪ Server end time

Assumption: similar request latency
 $T_3 - T_1 = T_4 - T_2$

▪ Report hit time: T₄

Measurement: AJAX performance

- Similar to full page load
- T₁ can be obtained by Javascript at client side most of the time;

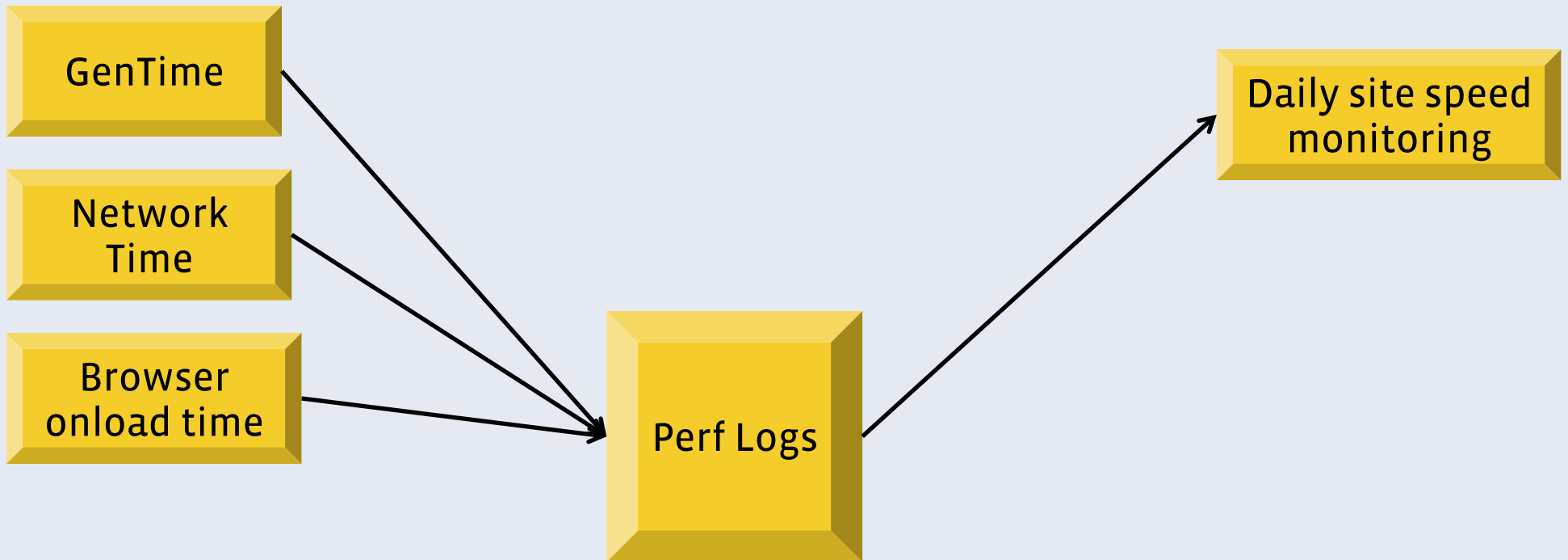
Extra difficulties:

- More than one AJAX actions can happen at the same time;
- Much more measurement data to deliver
- Definition of “meaningful” / “important” AJAX actions

Understanding: Day-to-day monitoring

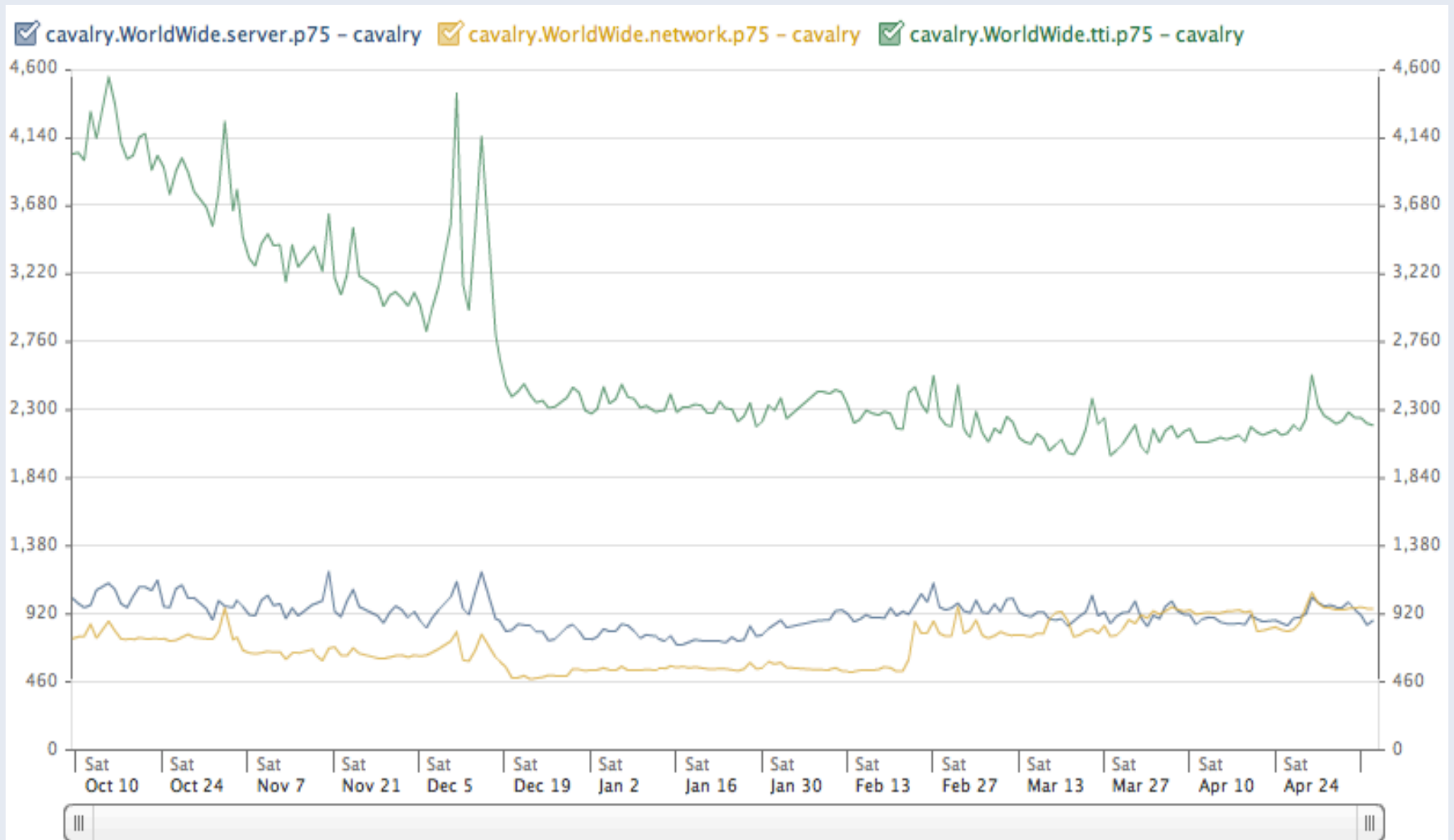
What's our speed?

- Collect gen time / network transfer time and render time



Understanding: Day-to-day monitoring

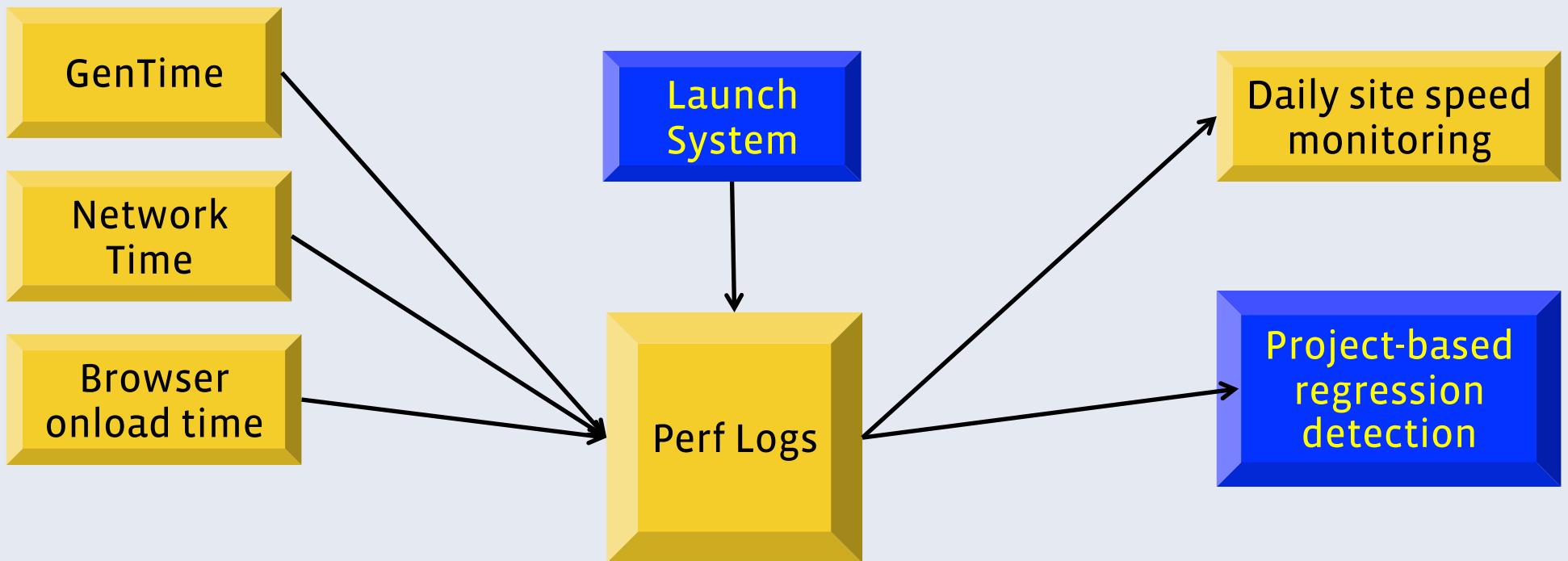
Example: TTI graph and its breakdown



Understanding: Project-based analysis

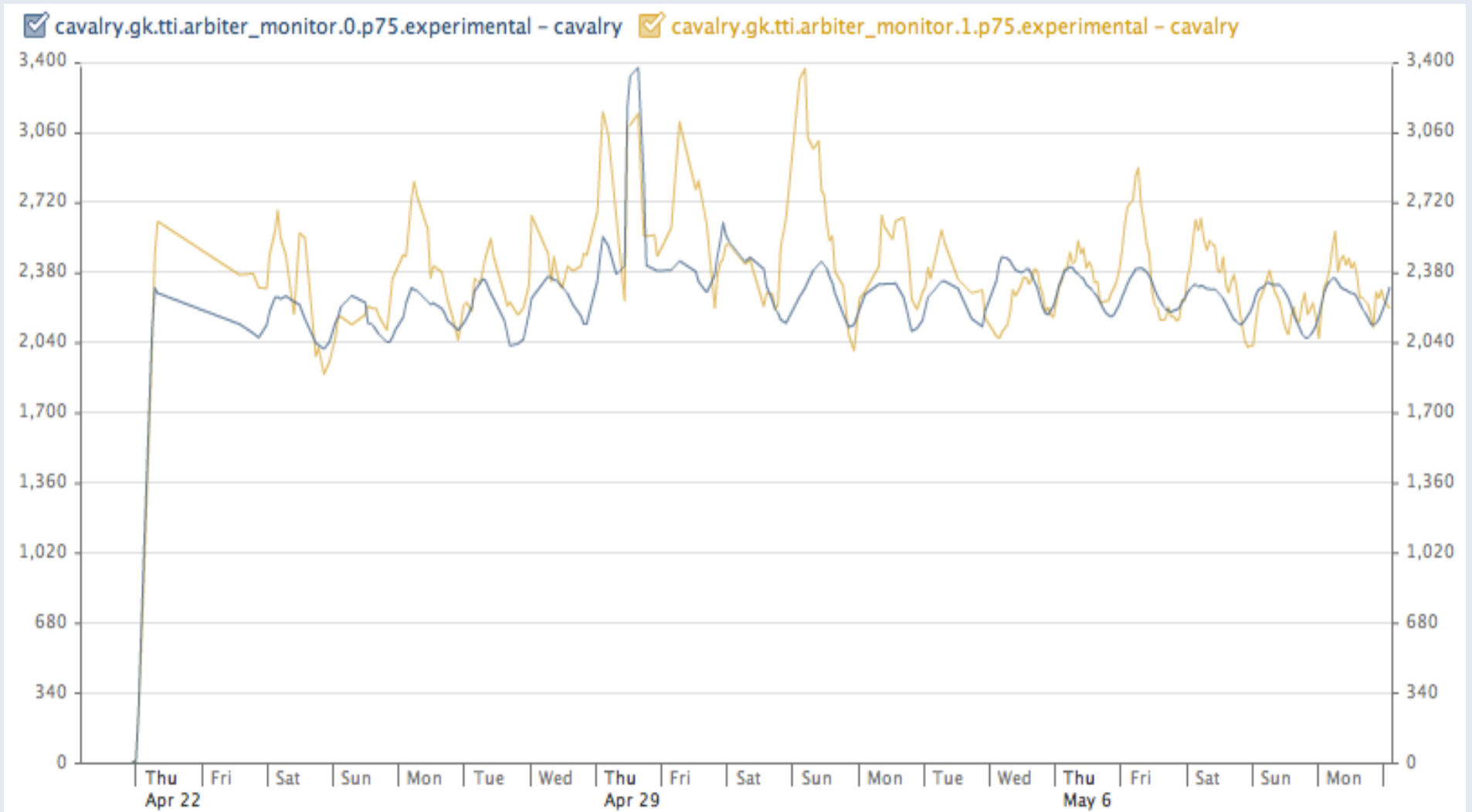
Who made it faster / slower?

- Integrated with Launch System: ~100 experimental launches



Understanding: Project-based analysis

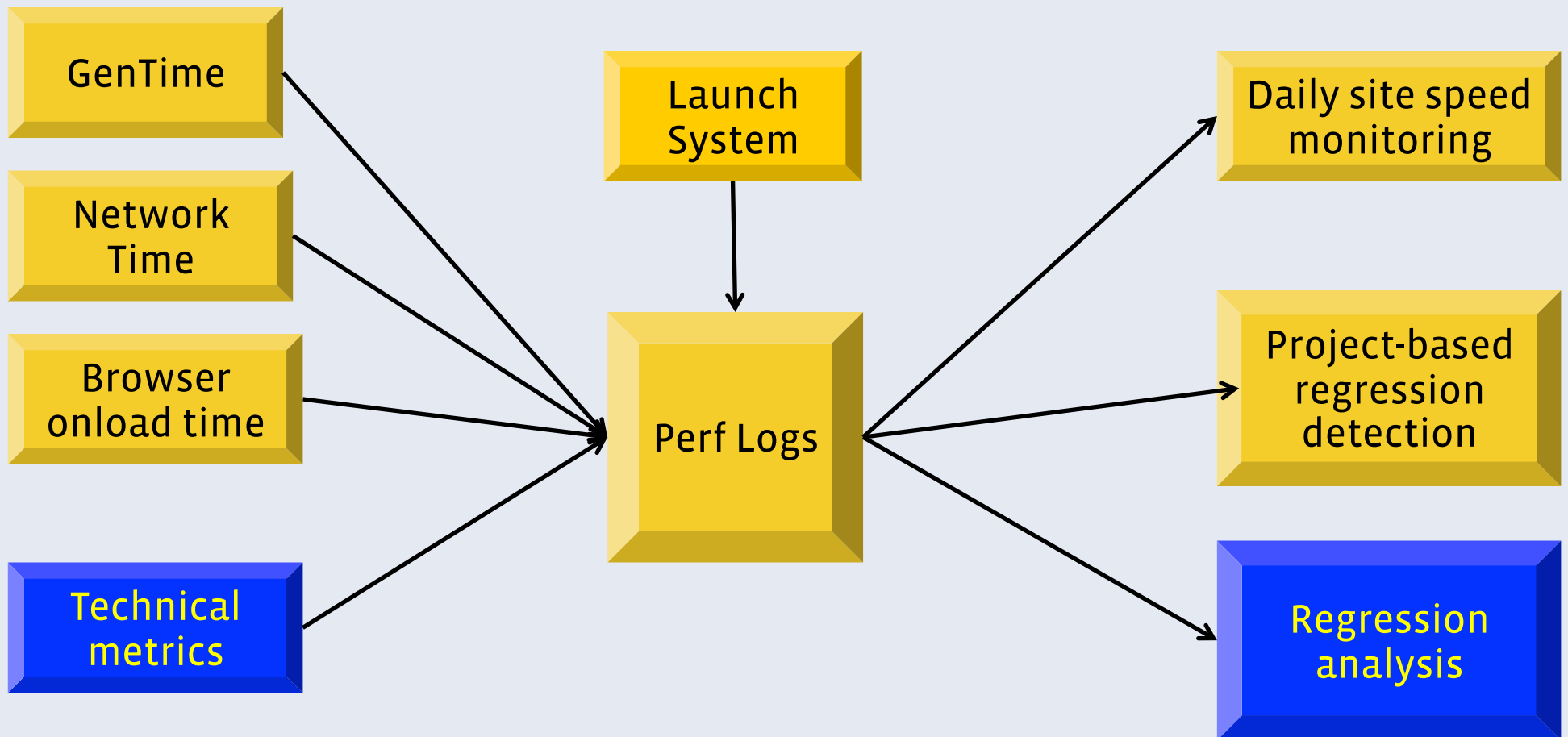
Example: A project launch's performance impact



Understanding: Numeric metrics

Why are we fast / slow? How can I fix it?

- Technical metrics

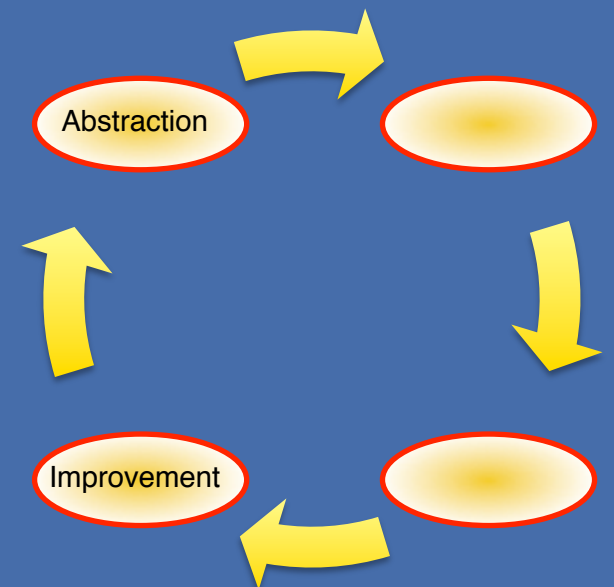


Understanding: Numeric metrics

Example: HTML bytes of home page hits



Empower the engineering team



Setting up goals

Areas that we work with product teams:

- Fighting regression
- Annual/Quarterly per product quality goals
 - Performance
 - reliability
 - Code quality
- New product design
 - Performance expectation
 - Code quality (usage of abstractions)

Providing convenient tools

Good tools to empower the product team:

- UI Component Library
 - Consistent user experience
 - Speedup product development
 - Greatly reduce CSS and HTML sizes
- Pagelet Gallery: Per pagelet performance analysis
- XHProf: PHP latency and CPU time drilldown analysis (Open sourced)

Providing convenient tools

Example: UI Component Library

facebook Home Profile Friends Inbox 2 Ben Mathews Settings Logout Search


- Introduction
- Core
 - Data Table
 - Grid
 - Image
 - Image Block**
 - Left Right
 - Link
 - List
 - Pager
 - Selector
 - Text
 - Tooltip
- More
- Legacy

<ui:image-block>

Consists of three parts: an image on the left, arbitrary content on the right, and optional content floated right.

Attributes: `enum imagetype {'icon', 'small', 'medium'} = icon`
Children: `(:ui:image | :ui:link, any, any?)`
File: `/flib/ui/xhp/core/image_block.php`

Example 1 [View Source](#)



Example 2 [View Source](#)



Creating necessary processes

Necessary processes that can be helpful:

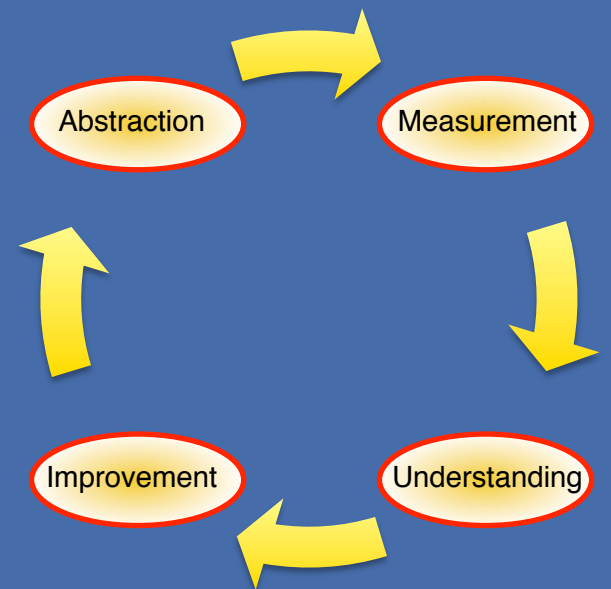
- New hire: Bootcamp / onboarding sessions
- New product: performance guideline
- Fire fighting vs development: Perf oncall
- Cross-team communication: “Perf point” -- “Perf adviser”

Creating necessary processes

Example: Bootcamp

- Each engineering employee, from fresh undergrads to highly experienced engineering directors, spends 6 weeks on Bootcamp
- Two onboarding courses about Web Performance
 - Basic tools (XHProf and etc)
 - Basic infrastructure (Static resource management, Pagelets, and etc)

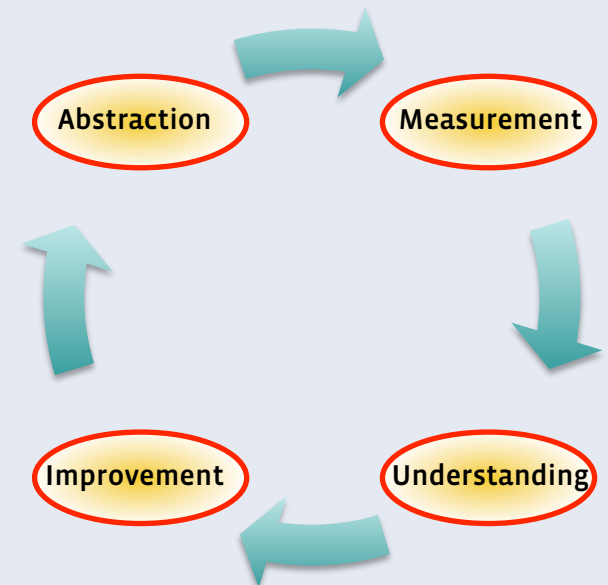
Summary



Achieving “Fast by default”

Making a large scale web site sustainably fast

- Focus on abstracts
- Data driven
- Empowering the whole engineering team





Thank you!

Xiaoliang “David” Wei
Facebook Inc.
www.facebook.com/DavidWei
DavidWei@acm.org

Velocity China, Dec 8th, 2010, Beijing