Bing Agility

MODERN ENGINEERING PRINCIPLES FOR LARGE SCALE TEAMS AND SERVICES

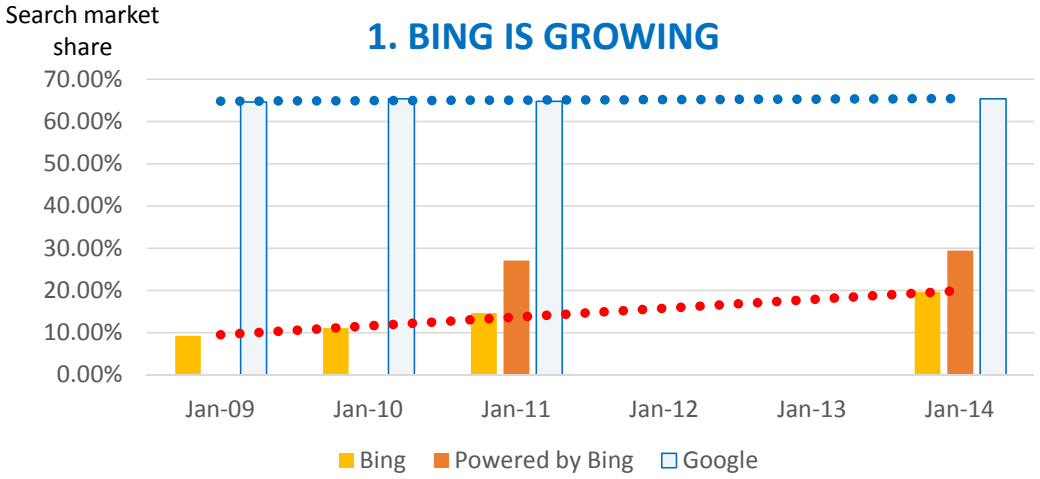
Outline

1. A bit about Bing
2. Velocity… What does it mean?
3. What is tested?
4. Modern Engineering Principles
5. The inner and outer loop
6. Performance gating

A bit about Bing

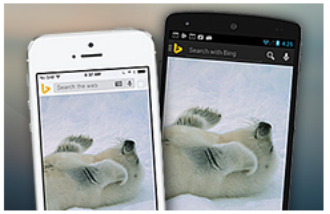
WW > 300M users, 9B searches/month
US >100M users, 4B searches/month

1. BING IS GROWING



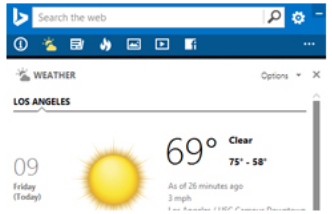
2. MORE WORK TO DO

Queries/UU (Dec 2014)	
Bing	38.3
Google	66.9



Bing on the road

Bing is designed to work beautifully across all your devices.



Bing Desktop

Get the beauty of the Bing homepage on your PC desktop every day.

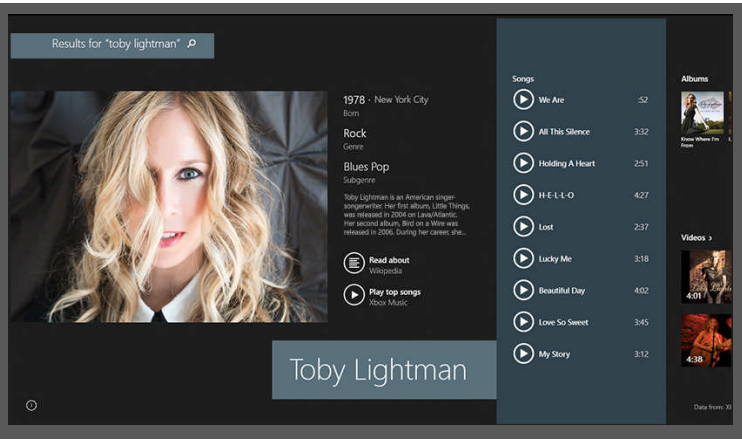
3. DIFFERENTIATE

Bing Homepage

Explore breathtaking images and download all your favorites.

Bing in the Classroom

Support digital literacy with ad-free search, free Surfaces, and lesson plans.



Bing makes predictions

Bing uses search, social, and other relevant data to make intelligent predictions about upcoming events, like sports games, reality TV shows, and more.

Velocity

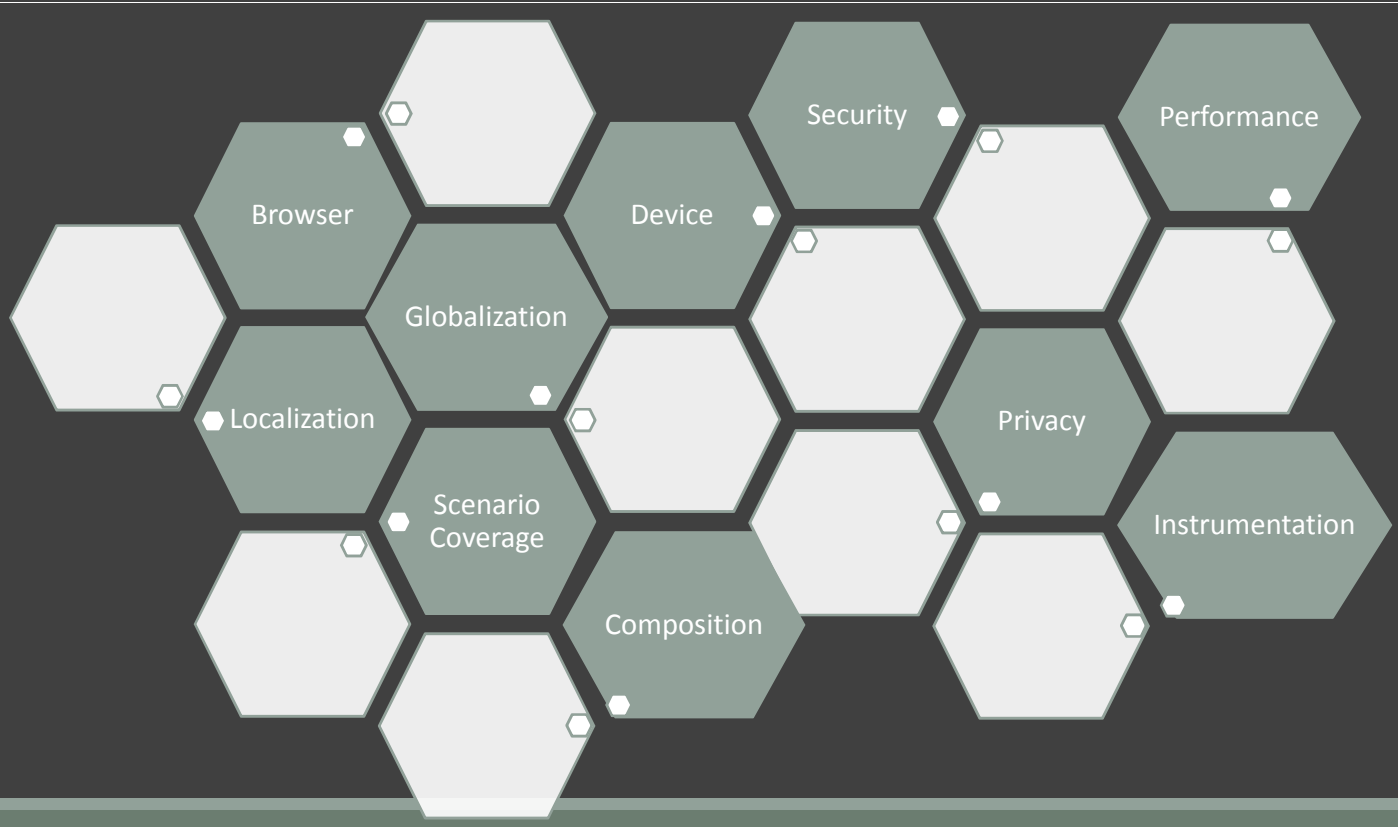
Does not mean…

- Shipping untested code… (any bozo can do that)

Does mean…

- Shipping thoroughly tested code…
- Shipping with high quality
- Shipping fast!

What is tested?



Modern Engineering Principles

Current engineering landscape

Hundreds of engineers

- *2000 engineers, across all continents*

Ship 4x/day

- Full build shipped to production, no live site issues!

Agile

- {design, dev, test} → ship (no P0 bugs) → repeat

One source tree

- Componentization, contracts, modularization

19.7% search market share (>30% share if Yahoo! is included)

Modern Engineering Principles

Test-Driven Evolution: 11 Principles

1. Automate every test, but don't test everything
2. Run all tests for every single check-in
3. Tests are binary: either they all pass, or they all fail
4. No test selection. Run them all. Scale thru HW + SW + Quota
5. Retire/Change old definitions and concepts
6. Embrace the Open-Source!
7. Testing in Production (deploy to production, test in production)
8. Deployment gated by tests: if any test fails, rollback
9. Defensive coding techniques (code + test case for every check-in, small check-ins, code behind flights, etc.)
10. Be truly data driven
11. Live Site remains the King!

1. Automate every test, but don't test everything

Make every test reliable:

- Use mock data to isolate the code
- Write Once, Run Against Multiple Contexts
- Have "contractual" tests running to validate FE ←→ BE schema

Trust modern tools:

- UI automation is no longer fragile (Selenium)
- Cloud helps with elasticity for your tests (scaling out)

Have a browser matrix, stick with it and deal with the rest!

2. Run all tests for every single check-in

Integration of tests with Code Flow

- Takes one hour for the first review to come (idle time)
- Changes → build → deploy → tests

20,000 tests <= 20min, code coverage ~65%

- Fast: mocked data
- Fast: Machines + Parallelism
- Fast: time quota system per feature team

3. Tests are binary: either they all pass, or they all fail

No concept of priorities until the test fails

All tests must pass, otherwise check-in's blocked

4. No test selection. Run them all. Scale thru HW + SW + Quota

The problems with test selection:

- A complicated imperfect system b/w product and tests
- Makes the process non-deterministic
- Some tests will rarely run!

“Throw machines at the problem!”

- This is what most big software corporations do
- Combination of HW + SW + Quota system

5. Retire/Change old definitions and concepts – Simplify!

Dev Documents and Test Plans ➔ One Page	Test case priorities ➔ Until they fail, they are P0	Test suites ➔ one	Test pass ➔ done when the check-in goes thru
Test environments ➔ production	But what about destructive? ➔ production	Code coverage ➔ just one data point	Ship decision ➔ from managers to engineers, based on bugs
	Obsessed about bugs ➔ Obsessed about user impact	Line b/w dev and test ➔ blurred	

6. Embrace the Open-Source!

Don’t try to compete with them – join them

All our tools are now all based on open-source

- Selenium, WebPageTest, PhantomJS, JS libraries, and many others

The work involved:

- Streamline the approval process
- Plumbing & Stitching the tools to work on MS tech

7. Testing in Production (TiP)

The problems with test environments:	• Maintenance • Not representative • Infinite catch-up game
Use an “invisible” PROD environment	• Behind a non-rotate flight • Behind a VIP that can’t be accessed from outside corpnet
Look at issue patterns in PROD	• Instrument every single aspect of the code • Big data/machine learning/telemetry techniques
What about destructive tests?	• Do it in PROD! Failovers/Load/Switch off the power to a DC • Better found by you than by someone else!

8. Deployment gated by tests: if any test fails, rollback

xPing: our version of Gomez/Keynote:

- Simple HTTP Gets
- xPing+: complex web-based scenarios using Selenium
- Runs continuously, alerts based on availability threshold
- E2E (no mocking)

Canary deployment:

- Deploy to one DC
- “Observe” the xPing tests
- All passed after N minutes? Push to the other DCs
- No? Rollback!

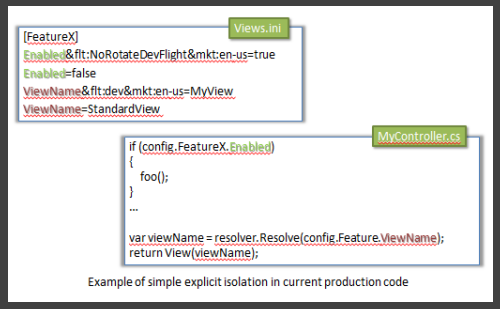
9. Defensive coding techniques

Code + functional test case for every check-in

Small, frequent check-ins

Defensive code – no assumptions!

Code behind a flight – switchable on/off:



Example of simple explicit isolation in current production code

10. Be truly data driven

- Instrument every aspect of your code
- Build a pipeline to gather and analyze the data
- Flight → Fail 90% → Learn → Ship 10%
- Make informed decisions based on data

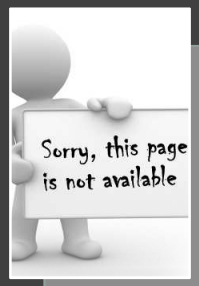
• Example:

Guardrail Metrics	Treatment	Control	Delta [%]	Pval
Quick Back 20	0.2295	0.2281	0.0014 [0.60%]	< 0.001
Algo Pane Load Time(Overall PLT)	1212	1208	4.055 [0.34%]	< 0.001
Revenue /UU	1.088	1.075	0.0130 [1.21%]	< 0.001
Truncated Revenue / UU	0.8571	0.8504	0.0067 [0.79%]	< 0.001
Distinct Queries / UU	14.67	14.67	–	1.001
Average Log Record Size (in KB)	111.4	111.1	0.2545 [0.23%]	< 0.001

11. Live Site

Heavy monitoring in production:

- Organic Monitoring (counters and rules)
- Synthetic Simple Monitoring (xPing, 10K tests)
- Synthetic Advanced Monitoring (exploratory)



Availability:

- Based on real traffic (*Search Merged Logs*)
- Real-Time

DRI – Designated Responsible Individual

ITR – Incident Tracking Record

Challenges & Learnings

- Management must embrace it
- Put dedicated engineers on the problems
- Be date-driven (things won't be perfect, but just do it!)
- This is a drastic change
  - Not everyone was happy… but don't try to please everyone!
- Have challenging and insane goals

The Inner Dev Loop (on demand)

Code Review Pipeline

BUILD
PASS
PASS

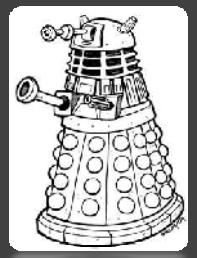
Build pipeline

BUILD

build

BUILD
PASS

Unit test



Check in Wizard

Verify test results

Source Code Repository

IXP

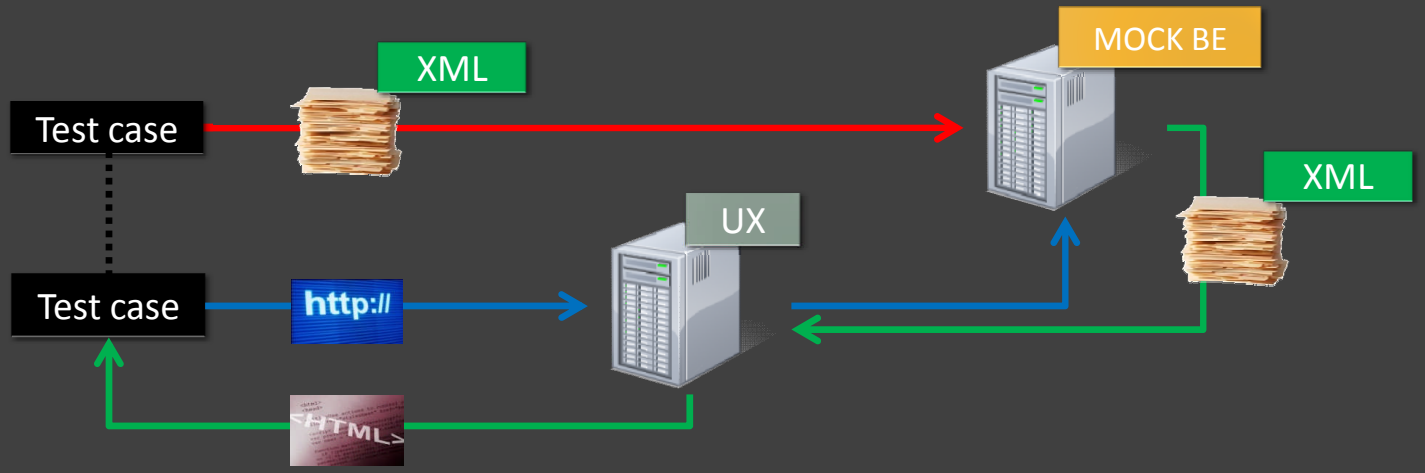
BUILD
PASS
PASS

Mainline tests

Bing UX Functional Automation

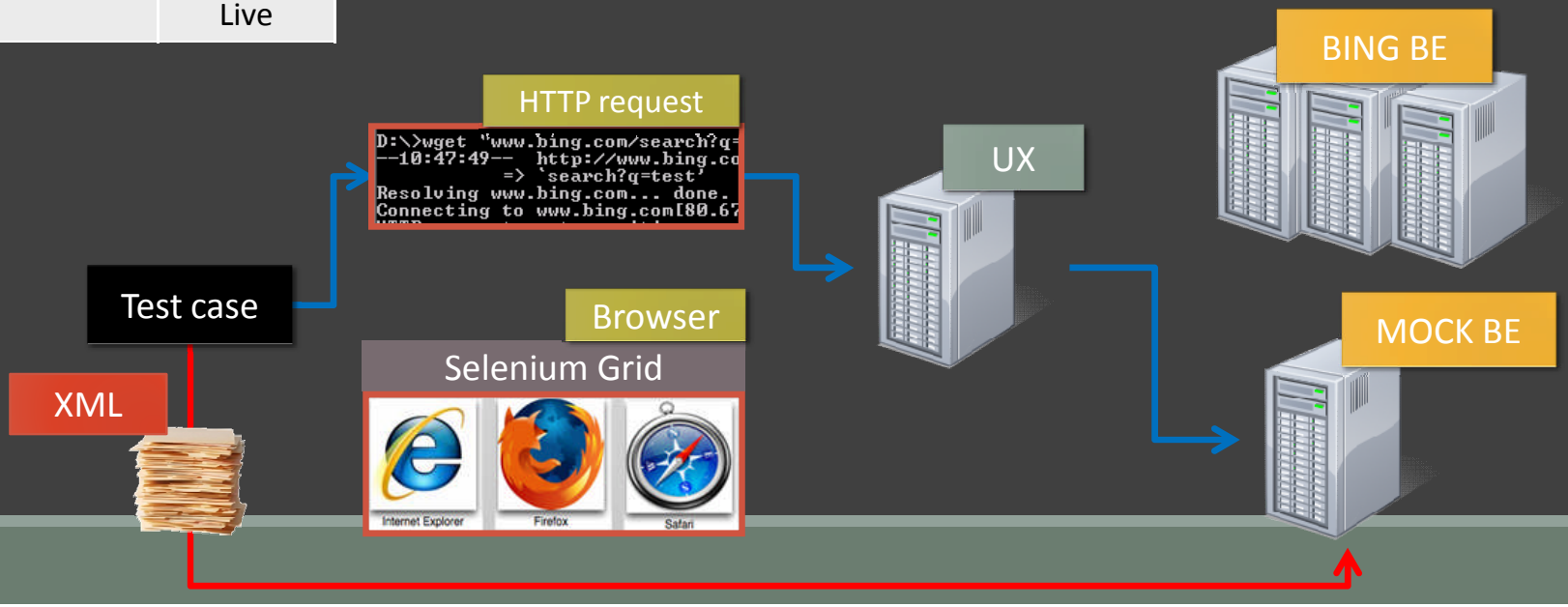
Mocked functional automation

- Create and deploy mocked data
- Request it as a Backend response



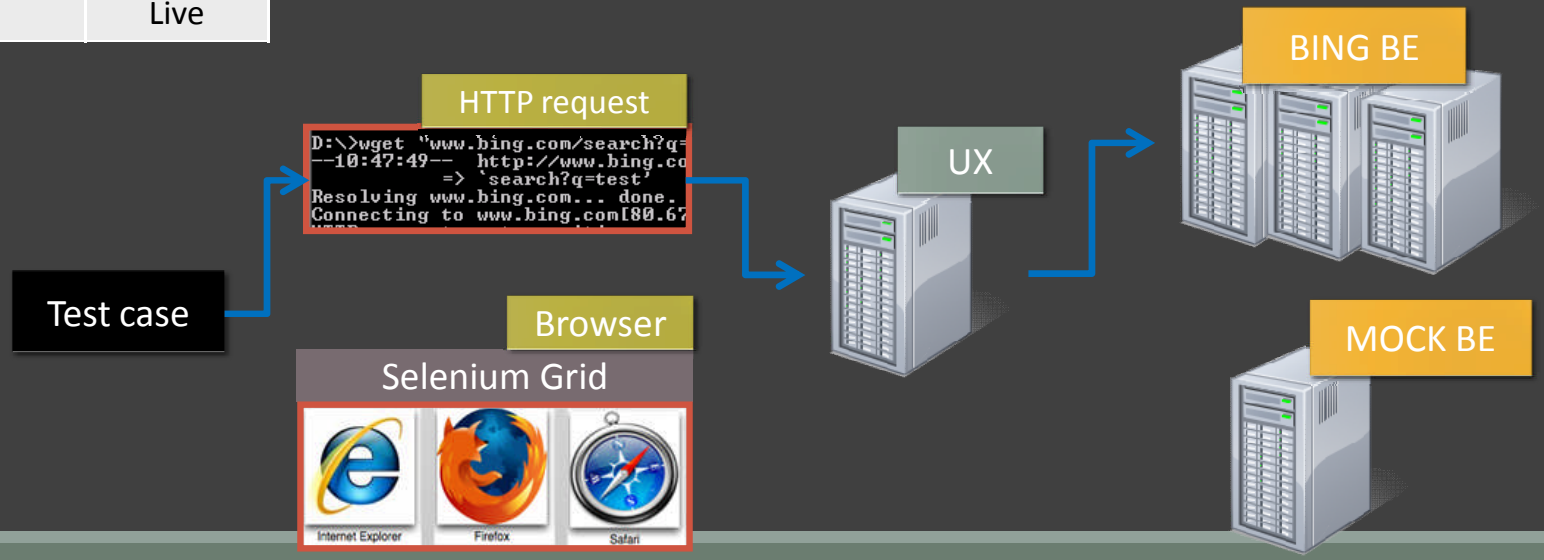
Bing UX Functional Automation

Vector	Data
HTTP request/response	Mock
HTTP request/response	Live
Browser-driven	Mock
Browser-driven	Live



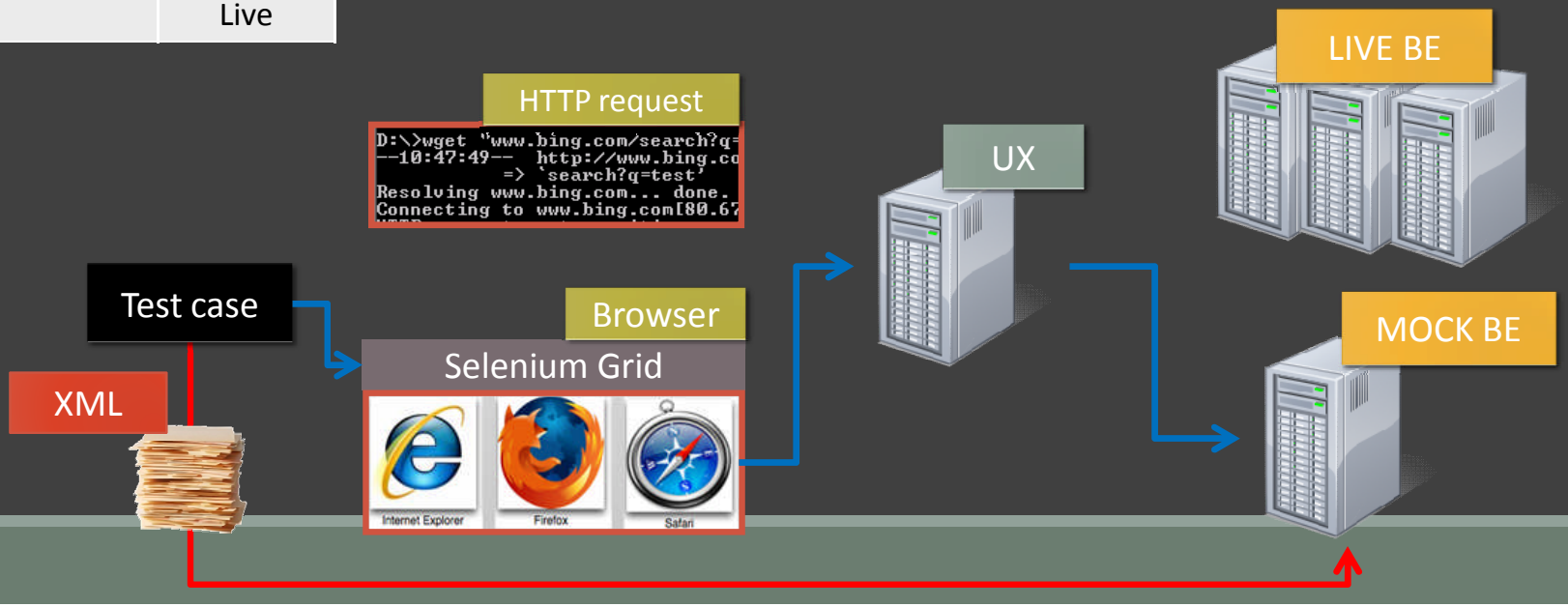
Bing UX Functional Automation

Vector	Data
HTTP request/response	Mock
HTTP request/response	Live
Browser-driven	Mock
Browser-driven	Live



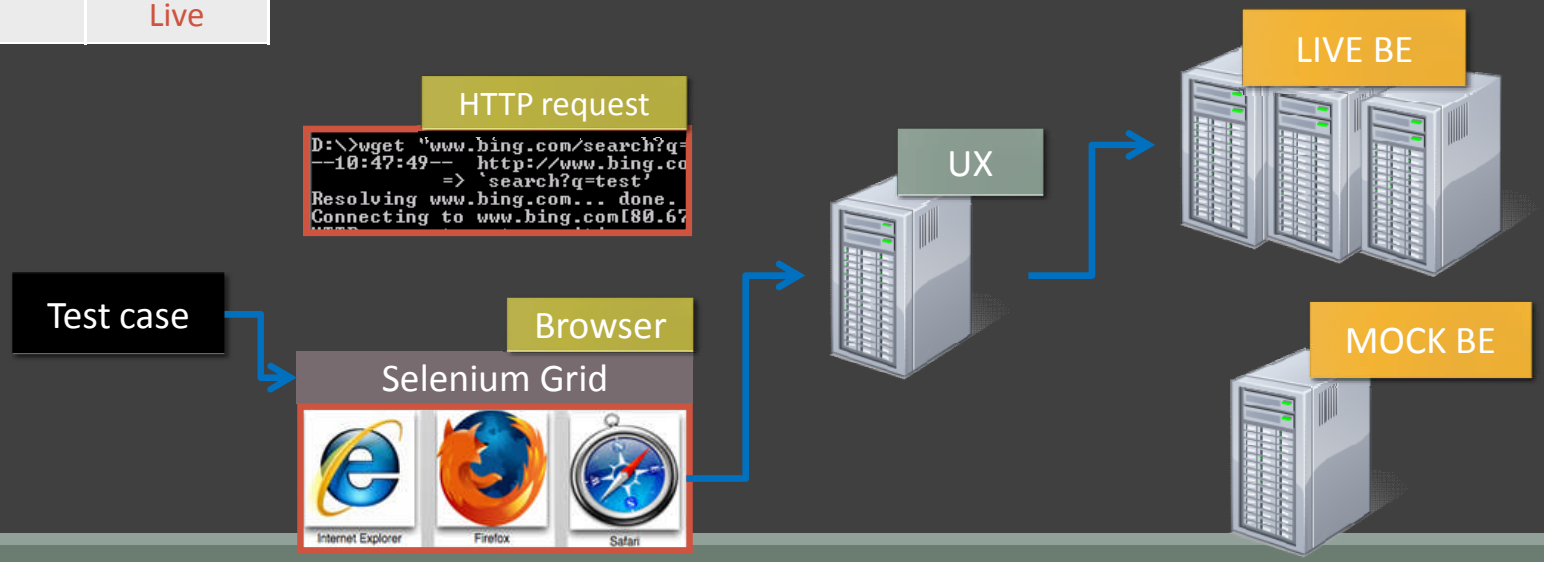
Bing UX Functional Automation

Vector	Data
HTTP request/response	Mock
HTTP request/response	Live
Browser-driven	Mock
Browser-driven	Live



Bing UX Functional Automation

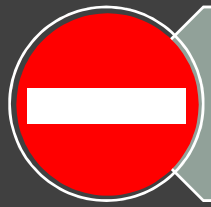
Vector	Data
HTTP request/response	Mock
HTTP request/response	Live
Browser-driven	Mock
Browser-driven	Live



Code Reviews

Parallel with build creation

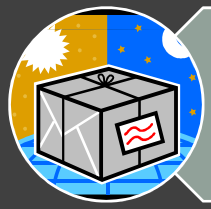
Parallel with test execution



Can block check-in…

Checked-in code

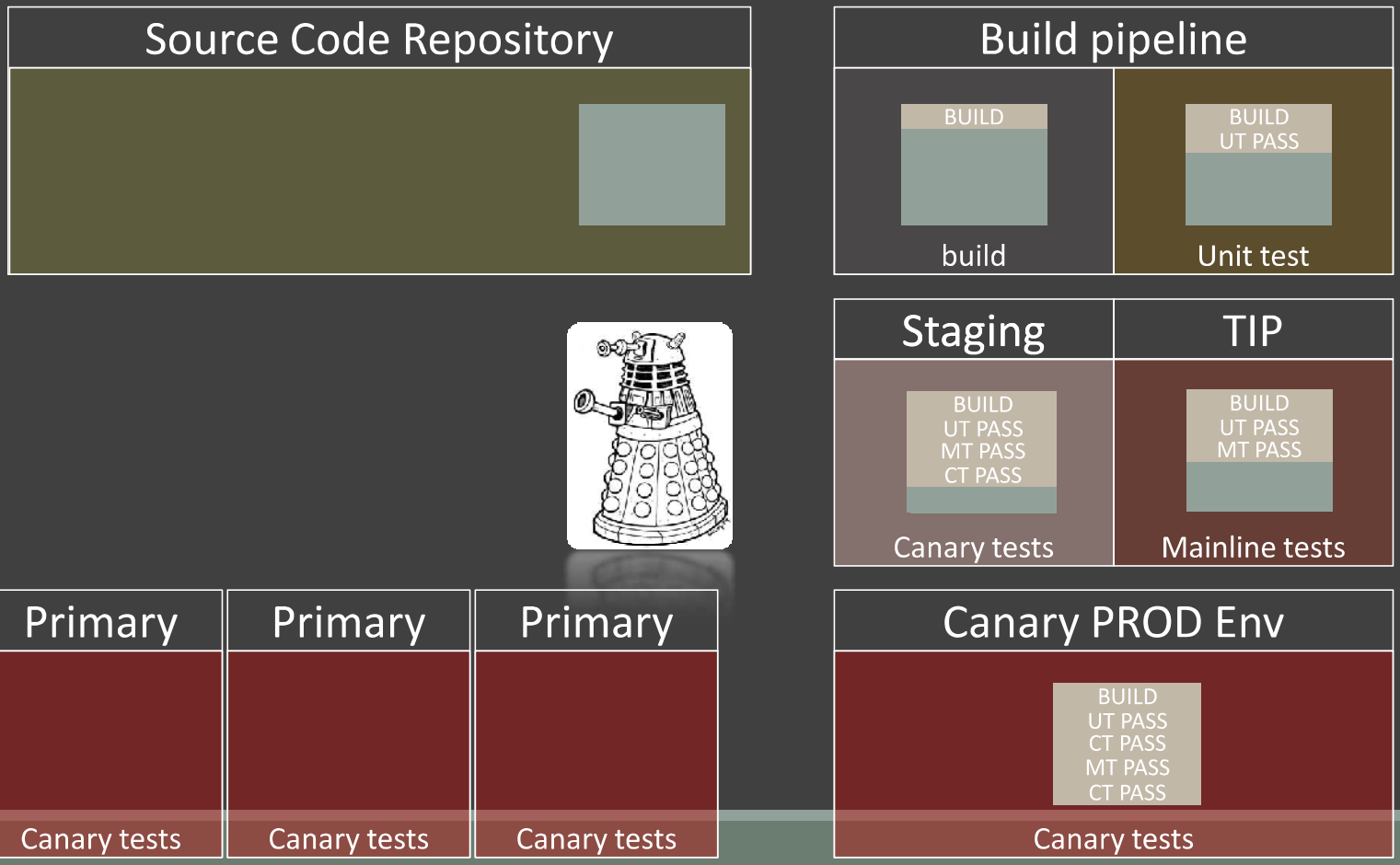
Has passed **ALL** tests



WILL ship within hours

OWNED by the feature teams

Continuous Delivery Loop (every day)



Performance Testing Strategy: Budgeting

Runs as a check-in test

Utilizes developer maintained budgets for resources

Below, identified an increase in page size due to a CSS change

```
89
    .b_focusTextExtraSmall,
90  .b_focusLabel,
91  .b_secondaryFocus,
92  .b_lText
93  {
94      font: 18px/normal 'Segoe UI', Arial, Helvetica, Sans-Serif;
95      line-height: 1.2em;
96  }
97
98  .b_focusTextExtraSmall
99  {
100     font: 18px/normal 'Segoe UI', Arial, Helvetica, Sans-Serif;
101     line-height: 1.3em;
102 }
103
104 .b_xlText
105 {
106     font: 24px/normal 'Segoe UI', Arial, Helvetica, Sans-Serif;
107     line-height: 1.2em;
108 }
109
110 .b_focusTextSmall,
111 .b_xxlText
112 {
113     font: 32px/normal 'Segoe UI Light', Arial, Helvetica, Sans-Serif;
114     line-height: 1.2em;
115 }
116
```

Current Status : TestsFa

Session Details

User: rihu
Session created: 5/12/2014 10:27:07 AM
Session completed: 5/12/2014 11:21:00 AM

Failed Tests

Name	#Pass	#Fail	#Skipped	#Total	JobState	Log	Duration
PPCWorking.xml	0	1	0	1	Completed	Log	00:43

All Tests

Name	#Pass	#Fail	#Skipped	#Total	JobState	Log	Duration
WPBlue.Performance.L4.xml	6	0	0	6	Completed	Log	01:25
PageFramework.Flights.xml	4	0	0	4	Completed	Log	00:05

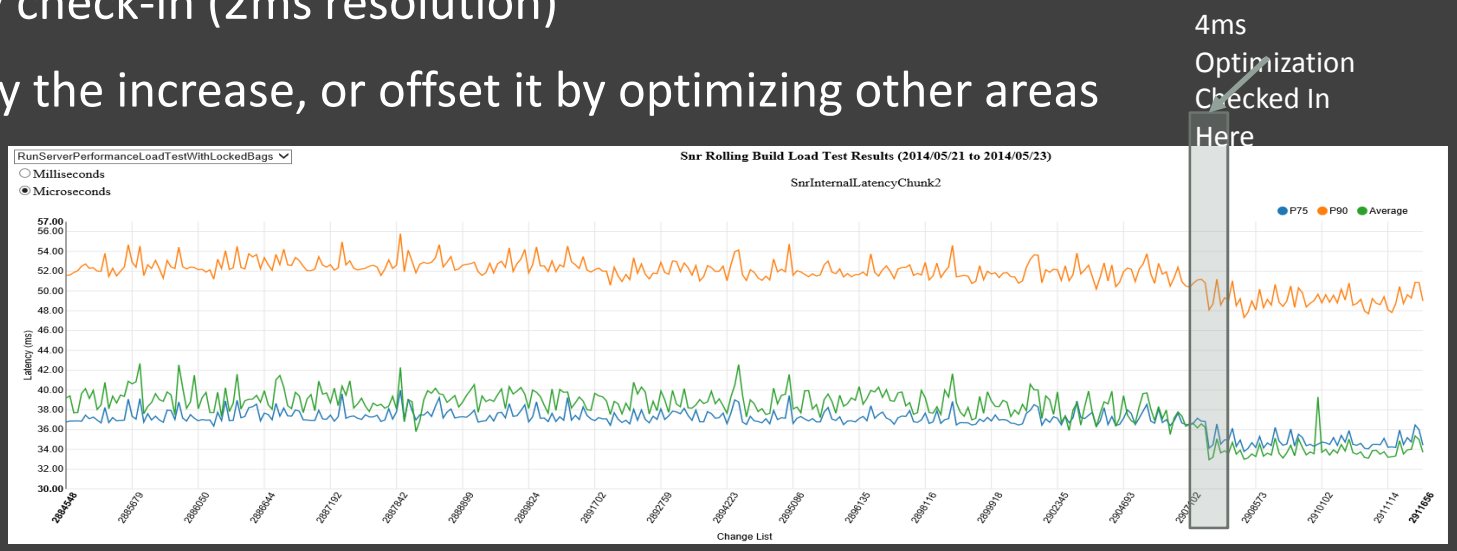
Performance Testing Strategy: Time (Load Test)

Forks traffic from production (no PII, ~1M queries)

Results from initial requests cached & replayed

Runs for every check-in (2ms resolution)

Options: justify the increase, or offset it by optimizing other areas



Questions?