UXtune – a toolkit to accelerate Android user interaction optimizations

Xiao-Feng Li
xiaofeng.li@gmail.com
Oct, 2011

Thanks to Ke Chen and Greg Zhu

Summary

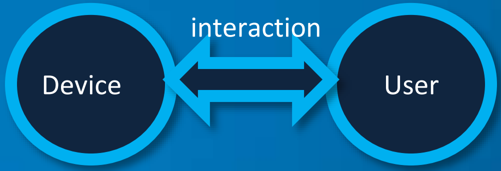
- UXtune is an engineering toolkit for Android user interaction analysis and optimization
- Tuning user interaction requires to understand the state transitions. We need,
  - Repeatable inputs to operate the device
  - Correlation of events between the analyzed entities
  - Metrics outputs to characterize the state transition

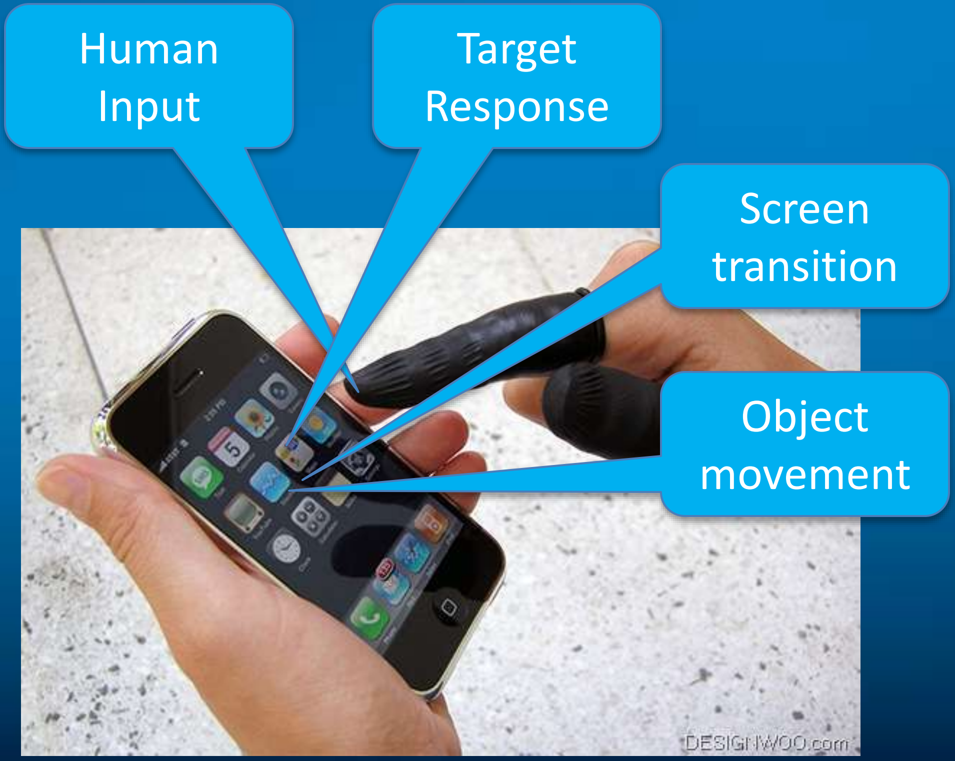
Agenda

- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

User Interactions with Client Device

- A sequence of interactions





- Does the input trigger the target **correctly**?
- Does the system act **responsively**?
- Does the graphics transition **smoothly**?
- Does the object move **coherently**?

Optimize User Interaction Systematically

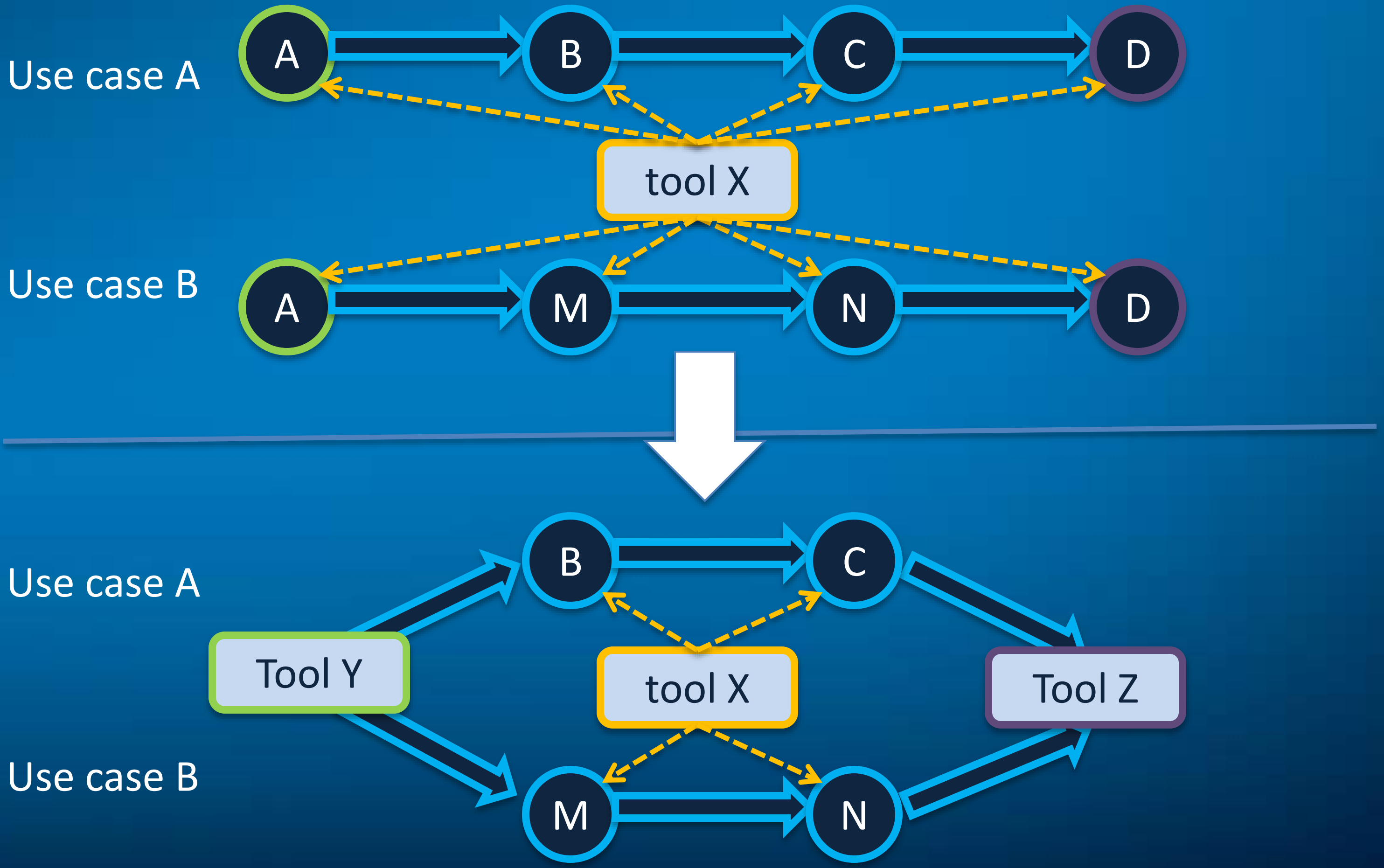
- What we need:
  - A well-established methodology
  - An engineering workload suite
  - An analysis/tuning toolkit
  - Sightings/requests/feedbacks from the users, etc.

- (The methodology details are in another deck)
- (The workload suite details are in another deck)

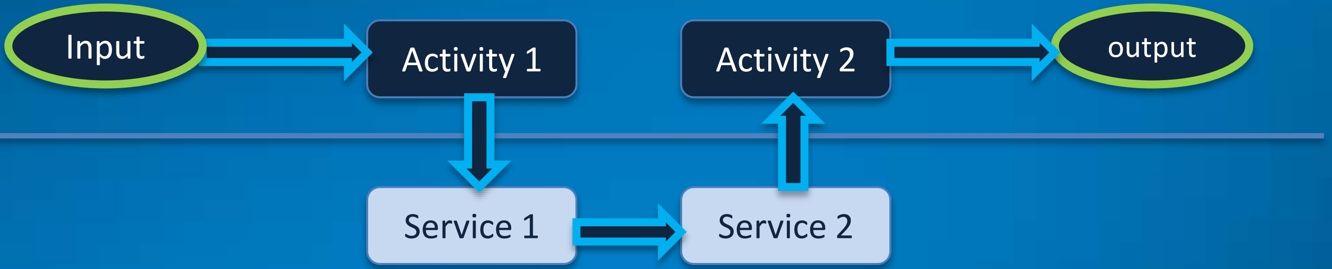
Relation Between Workloads and Toolkit (1)

- Workloads are to characterize the representative usage models of the system
  - One workload can execute part of the system
  - A comprehensive suite can cover most of the system
- Tools are to analyze the system
  - A tool itself does not represent a use case
  - A tool can be used to analyze a usage model
  - The common part of multiple usage models can be abstracted into a tool

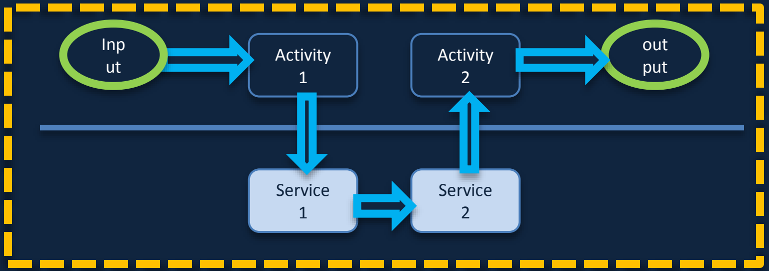
Relation Between Workloads and Toolkit (2)



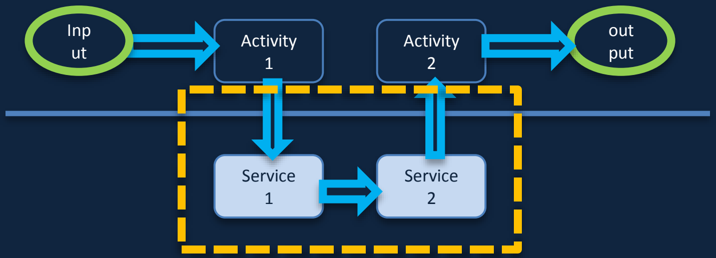
Relation Between Workloads and Toolkit (3)



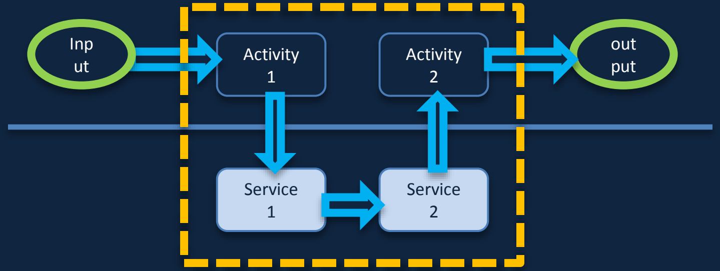
1. Standalone workload



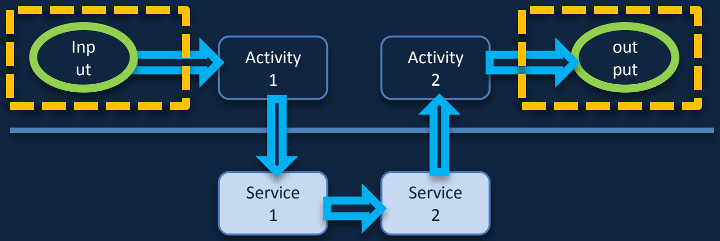
2. Micro workload



3. Measurement tool



4. Scenario driver



UXtune Toolkit

- To analyze and optimize Android, we need
  - Repeatable inputs operating the device
    - Android input-Gestures
  - Sequence of interaction events between the system components, such as event, frame, thread, etc.
    - Android UXtune
  - Metrics outputs characterizing the behavior
    - Android meter-FPS
    - Android app-launch
    - Android touch-pressure

Agenda

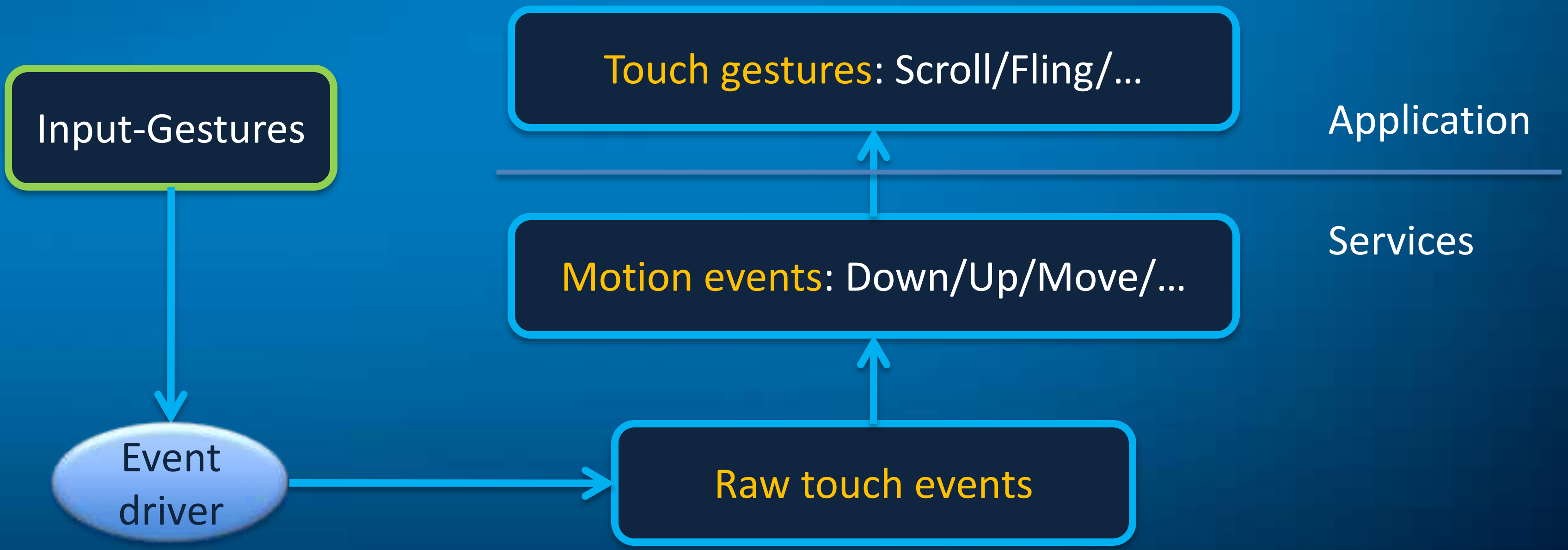
- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

Android Tool for Inputs: Input-Gestures

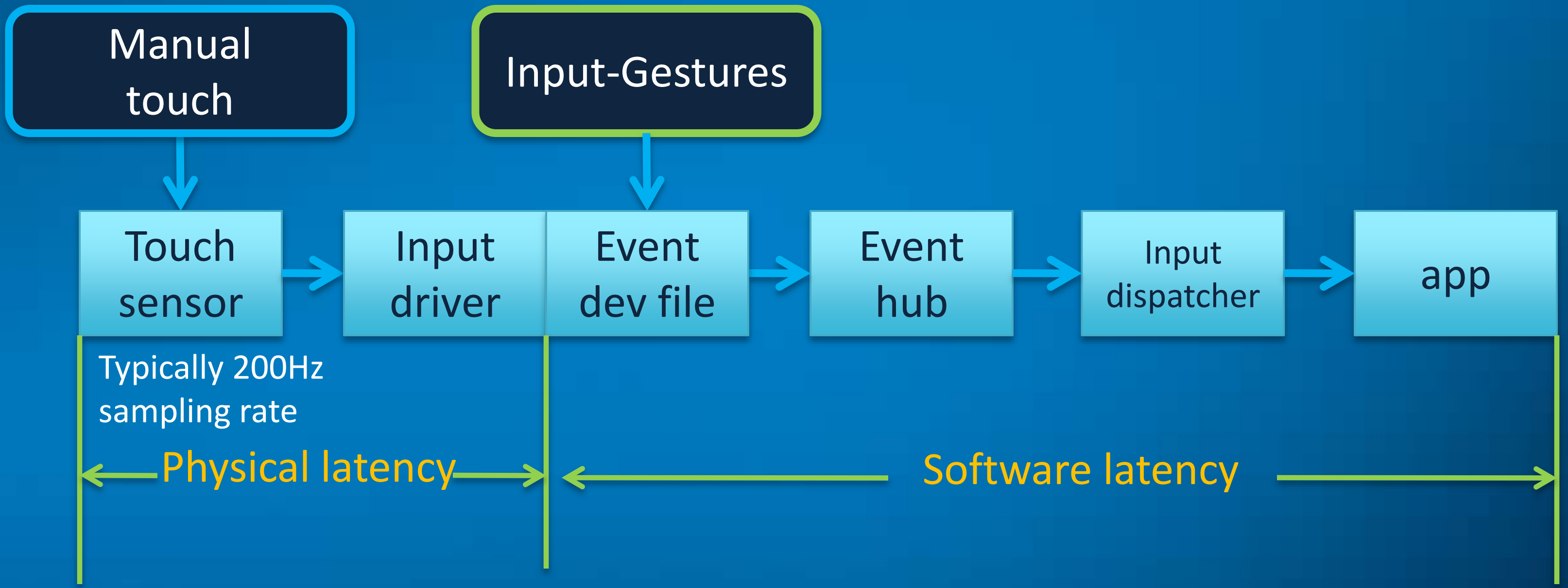
- A tool to generate *standard* touch gestures
  - So that people have same and repeatable inputs
- Supported gestures
  - Scroll: up/down/left/right from specified start position to specified end position in specified time
  - Fling: up/down/left/right at specified position
  - Zoom: in/out at specified position with specified span
  - Tap (double taps): at specified position
  - Long press: at specified position for specified duration

From Events to Gestures

- All gestures can be generated by Input-Gestures by emitting raw touch events



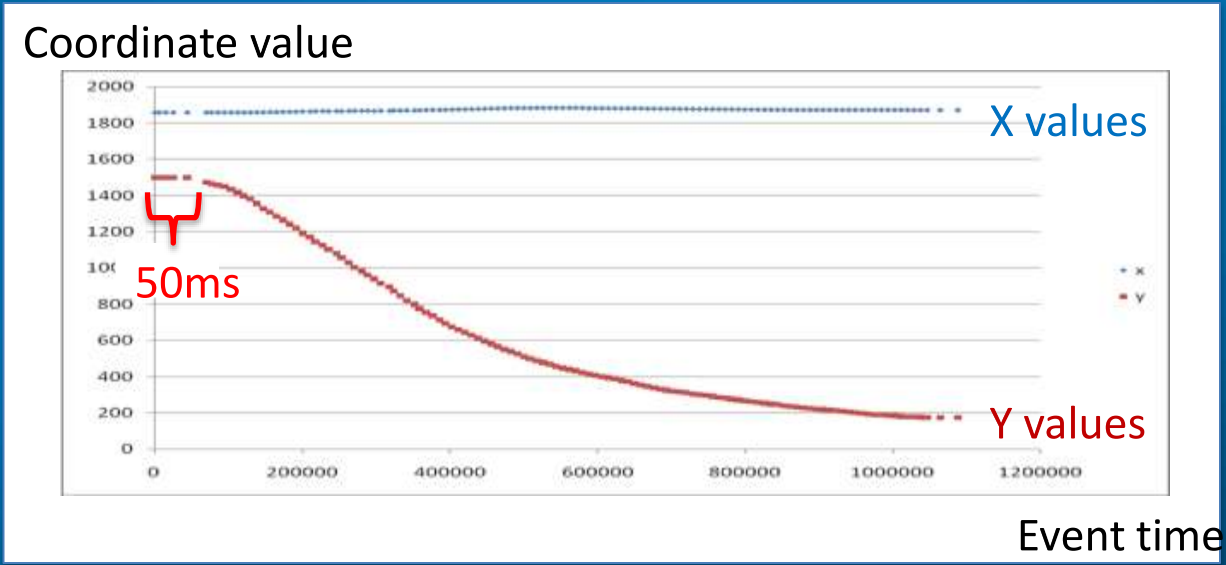
Input-Gestures vs. Manual Touch



- Software latency is our optimization focus
  - Software latency is around x100ms
  - Touch sampling rate is typically 200HZ (5ms interval)

Example: Scroll Gesture Generation (1)

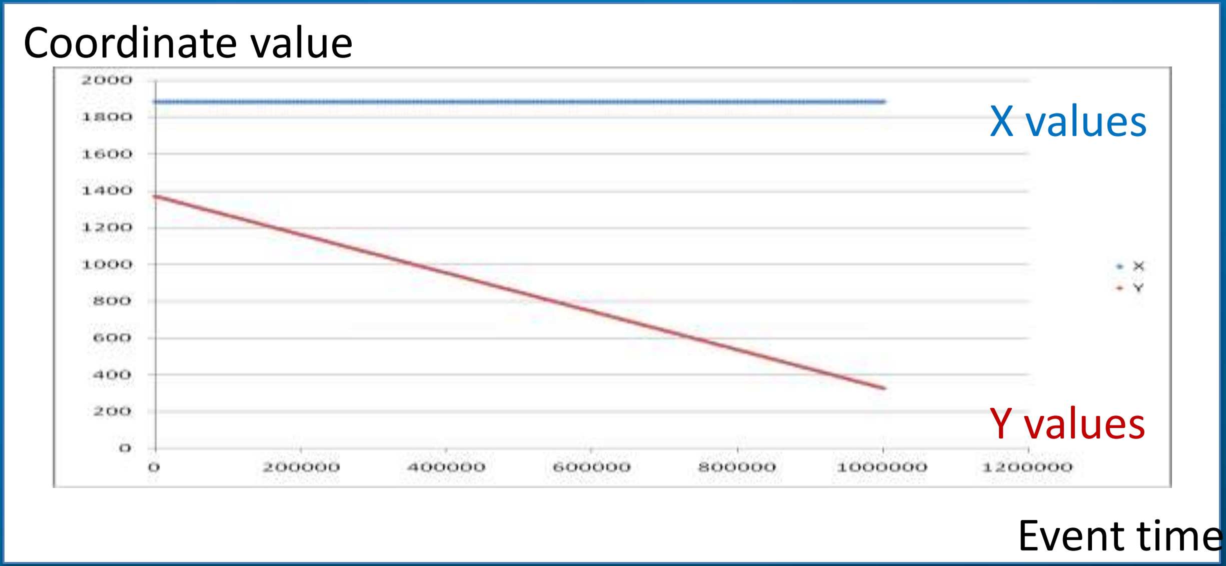
- A common recorded raw event sequence of a scroll



- The leading events stay at same position
  - In this example, the duration is 50ms
  - Should be removed for scroll response time measurement

Example: Scroll Gesture Generation (2)

- A simply generated raw event sequence of a scroll



- The leading events move faster than real sequence
  - Gesture detection identifies the “scroll” earlier than real
  - In this example, it shortens the response time by 10ms

Example: Scroll Gesture Generation (3)

- Ensure the generated gestures are comparable across different platforms
  - Across different resolutions, screen size
  - With different event format

Events of same gesture on Device X

```
1000000000 3 48 1
1000000010 3 53 3284
1000000020 3 54 2747
1000000030 0 2 0
1000000040 0 0 0
1000005000 3 48 1
1000005010 3 53 3284
1000005020 3 54 2735
```

Events of same gesture on Device Y

```
1000000000 3 48 1
1000000010 3 53 1810
1000000020 3 54 1515
1000000030 0 2 0
1000000040 0 0 0
1000005000 3 48 1
1000005010 3 53 1810
1000005020 3 54 1508
```

Agenda

- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

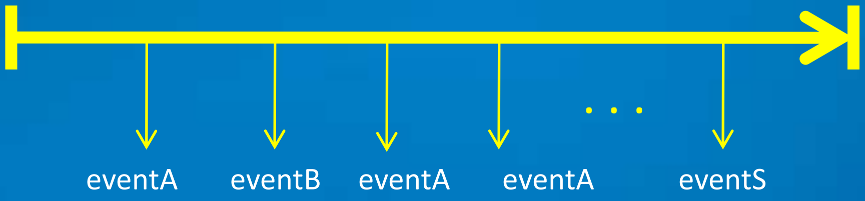
Android Tool for Analysis: UXtune

- A tool to assist the analysis of user interactions
  - The key is to characterize the state transitions
- UXtune design idea
  - Vertical correlation: Map system events across layers to user-level activities
    - E.g., Events, gestures, frames
  - Horizontal correlation: Correlate runtime activities between different system entities
    - E.g., a thread triggers a garbage collection
  - Visualization based on pyTimeChart

BKM in User Interaction Tuning

Traditional tuning BKM

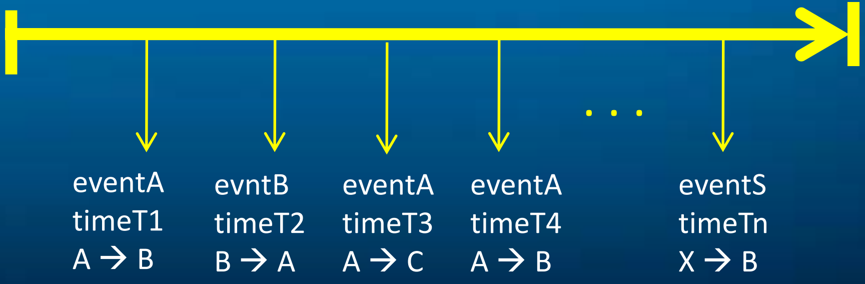
Inputs:
- System
- Workload
- Tool



Outputs summary:
- #Events happened
- #Events sampled
- Map to app code

User Interaction tuning BKM

Inputs:
- System
- Workload
- Tool



Outputs sequence:
- Events at T : X→Y
- Map to system entities
- Map to app actions

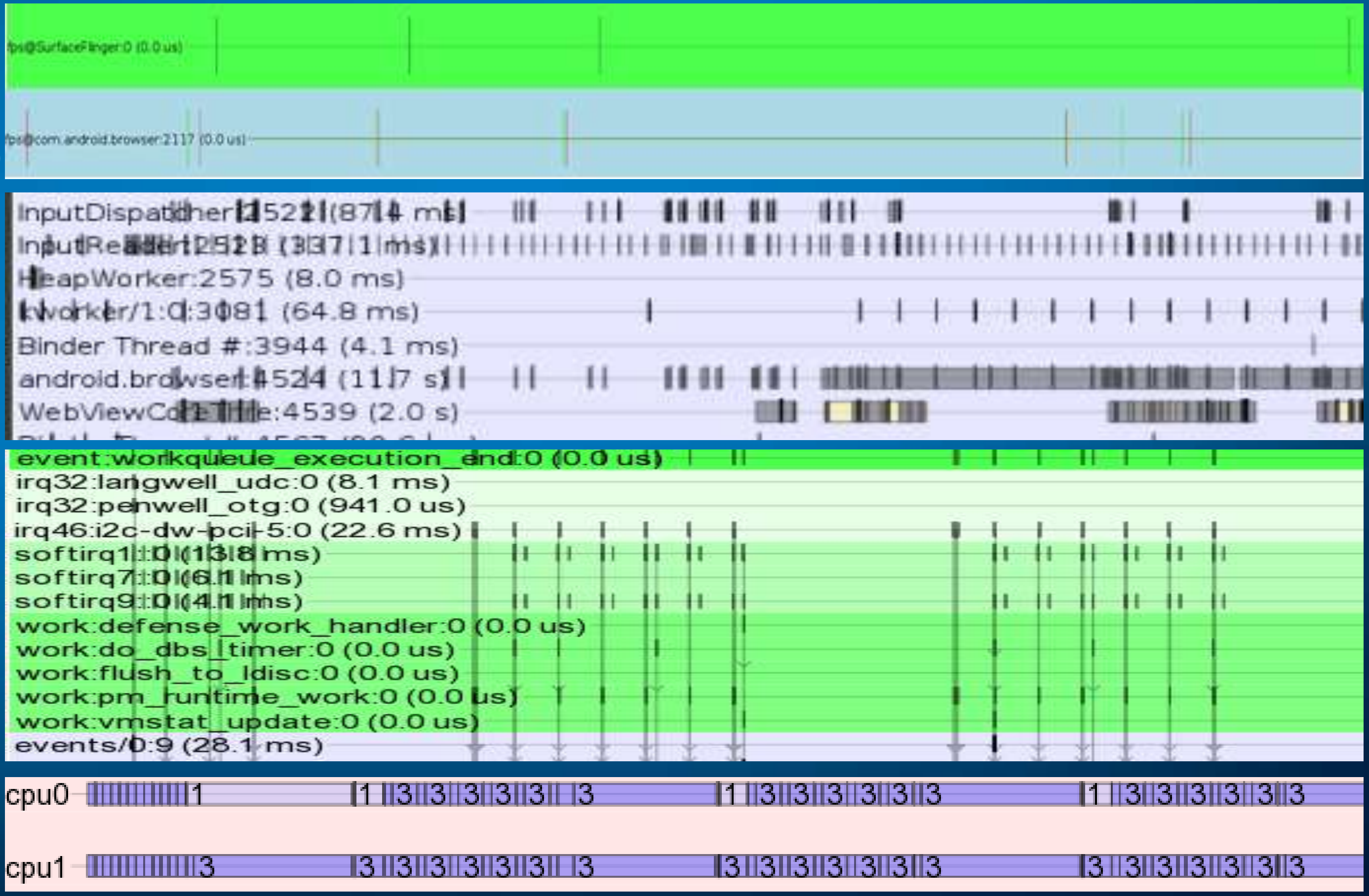
Visualize Vertical/Horizontal Correlations

User level activities (events, frames)

Android threads

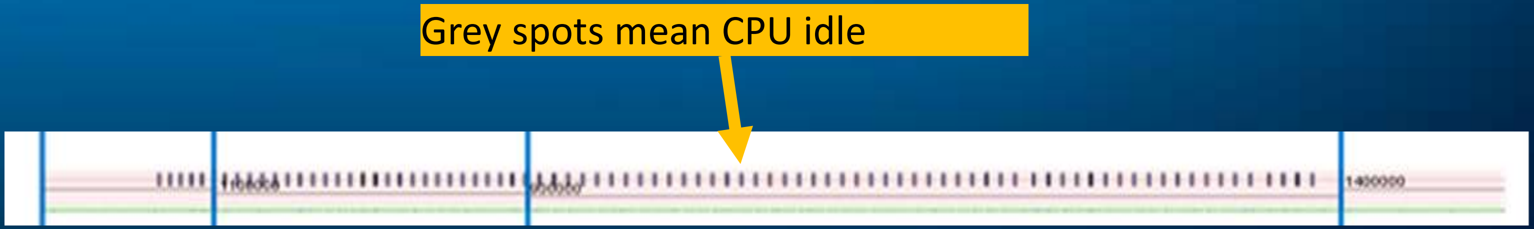
OS threads

CPU states


fps@SurfaceFlinger:0 (0.0 us)
fps@com.android.browser:2117 (0.0 us)
InputDispatcher:2521 (87.4 ms)
InputReader:2528 (337.1 ms)
HeapWorker:2575 (8.0 ms)
kworker/1:0:3081 (64.8 ms)
Binder Thread #:3944 (4.1 ms)
android.browser:4524 (11.7 s)
WebViewCoreThread:4539 (2.0 s)
event:workqueue_execution_end:0 (0.0 us)
irq32:langwell_udc:0 (8.1 ms)
irq32:penwell_otg:0 (941.0 us)
irq46:i2c-dw-pci-5:0 (22.6 ms)
softirq1:0 (13.8 ms)
softirq7:0 (6.1 ms)
softirq9:0 (4.1 ms)
work:defense_work_handler:0 (0.0 us)
work:do_dbs_timer:0 (0.0 us)
work:flush_to_ldisc:0 (0.0 us)
work:pm_runtime_work:0 (0.0 us)
work:vmstat_update:0 (0.0 us)
events/0:9 (28.1 ms)
cpu0
cpu1

Example: UXtune Analyzes CaffeinMark

- A problematic period in CaffeinMark execution
  - The problematic period occupies about 20% in total execution time
  - The idle spots (CPU idle time) together take about 20% of the problematic period
  - Performance impact: 20% * 20% = 4%
    - Not mention the incurred CPU frequency adjustment

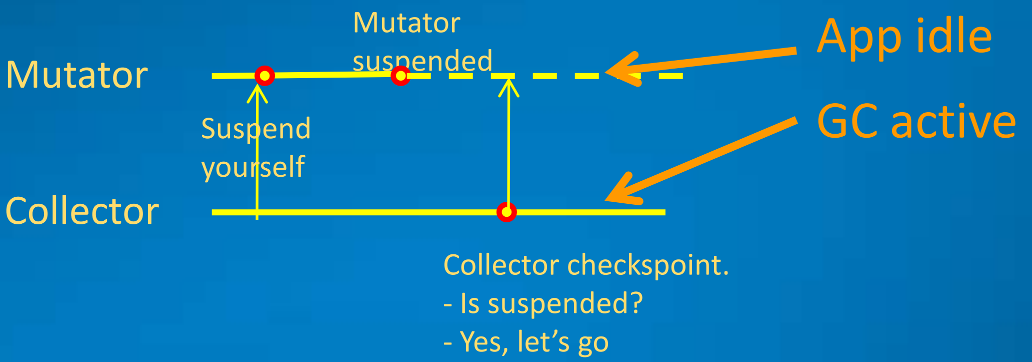


Android Concurrent GC Design

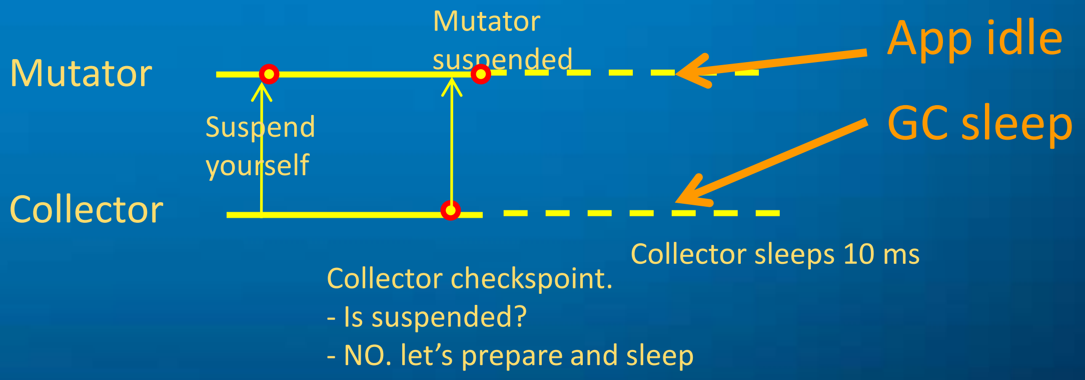
- GC design led to CPU idle because no active threads run
  - GC needs to pause app threads for root enumeration
    1. GC thread sets a flag asking app thread(s) to suspend for GC root set enumeration
    2. GC thread checks if app is suspended. If not yet, GC thread yields to let app run to suspend
    3. GC thread comes back to check again. If not, GC thread sleeps for 10ms
    4. App is suspended at some time point (possible CPU idle)
    5. GC thread wakes up, finishes root enumeration, and lets app resume

Interactions Between GC thread and App

Good scenario



Bad scenario

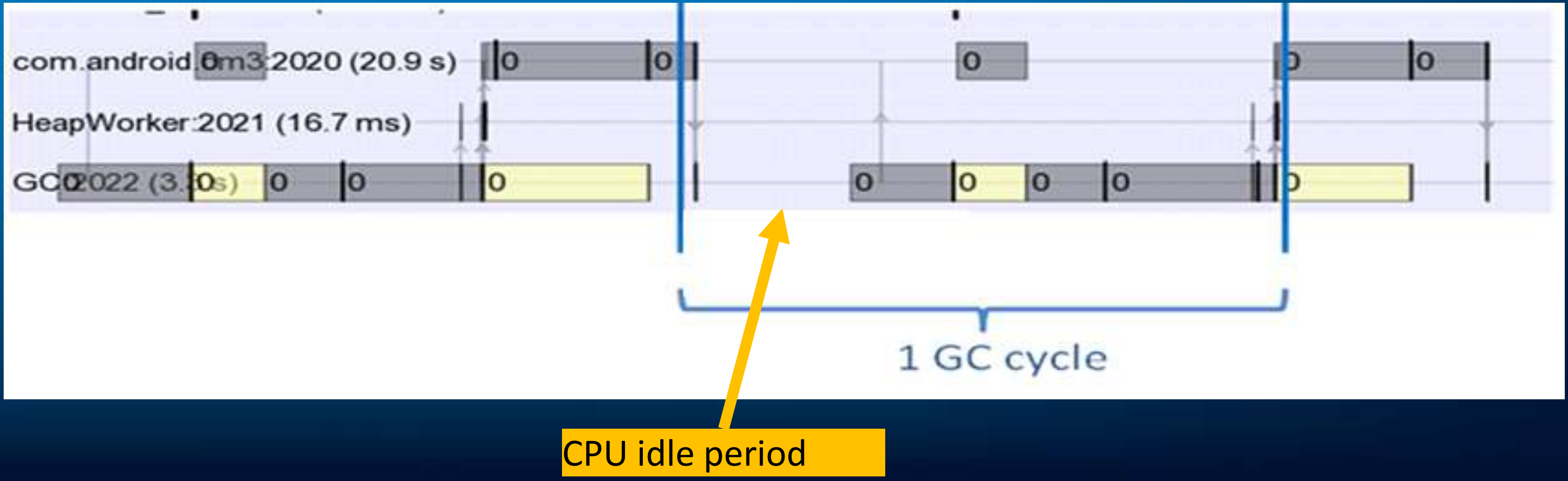


In the bad scenario

- No active thread for a moment

More UXtune Analysis with CaffeinMark

- The execution mainly involves two threads
  - CaffeinMark app thread (com.android.cm3)
  - GC thread
- Both are idle for a moment
  - Bad GC scenario happens in CaffeinMark

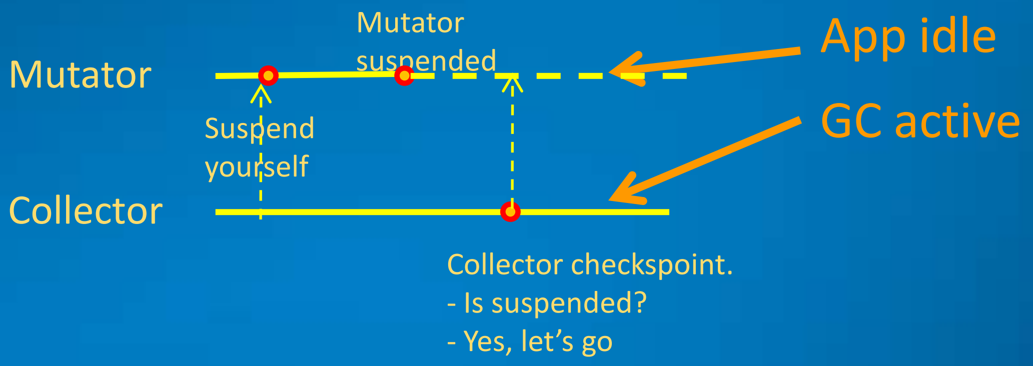


Optimizing Android Concurrent GC

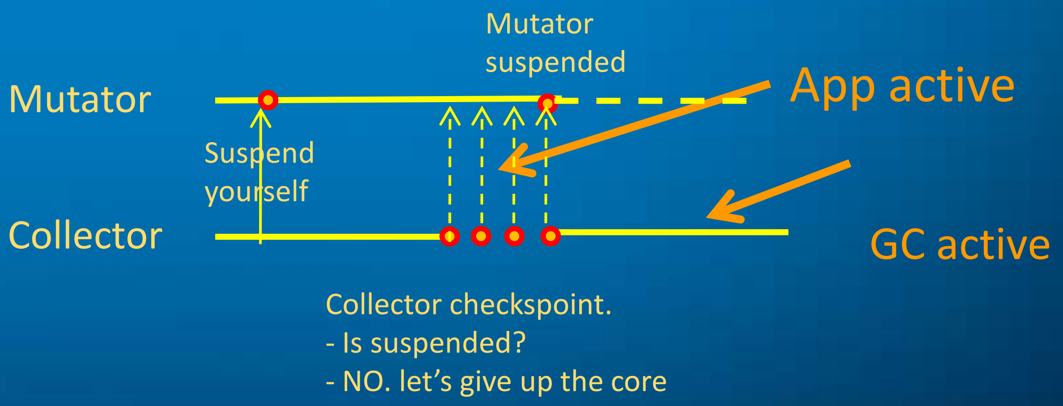
- In the GC bad scenario, replace the 10ms sleeping with a CPU-yielding action
  - GC thread gives up the core instead of sleeping
    1. GC thread notifies the app thread(s) to suspend
    2. GC thread checks if app is suspended. If not yet, goto Step 3. If yes, goto Step 4
    3. GC thread yields to let app run to suspend. When GC thread comes back, goto Step 2
    4. App is suspended at some time point. If no other thread, the GC thread should be scheduled to run
    5. GC thread finishes root enumeration, then lets app resume. GC thread continues collection concurrently

GC Scenarios with Improvement

Good scenario
Unchanged



Bad scenario
Improved



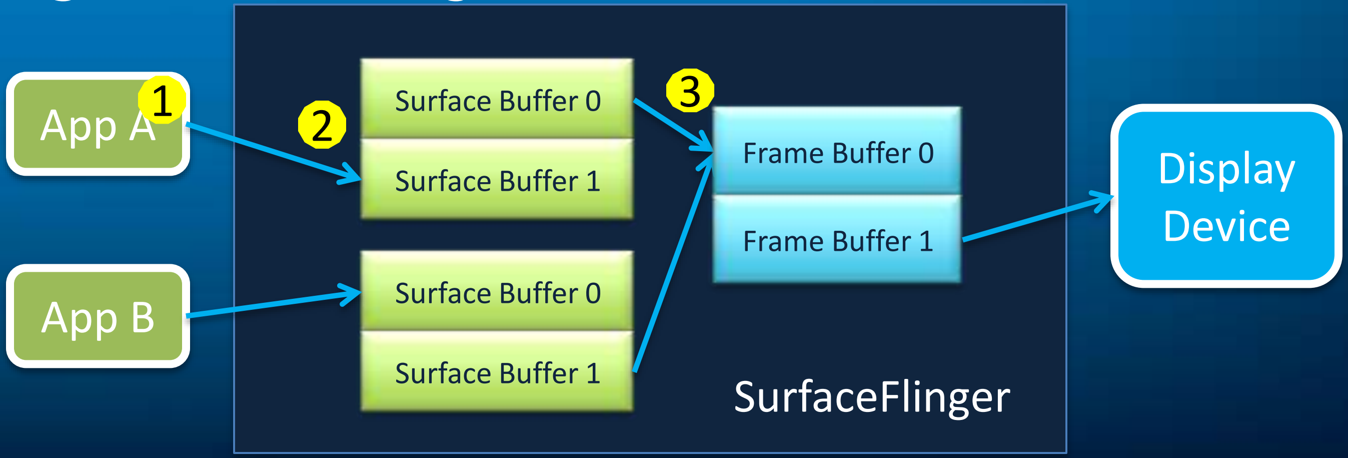
In the bad scenario
- Active thread always existing

Agenda

- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

Android Tool for Metrics: Meter-FPS

- A tool to measure the FPS value of the system
  - Include other metrics like maximal frame time, frame time variance, #long-time-frames, frame drop rate
- Design idea
  - Intercept the graphics processing paths to get the logs of the every frame

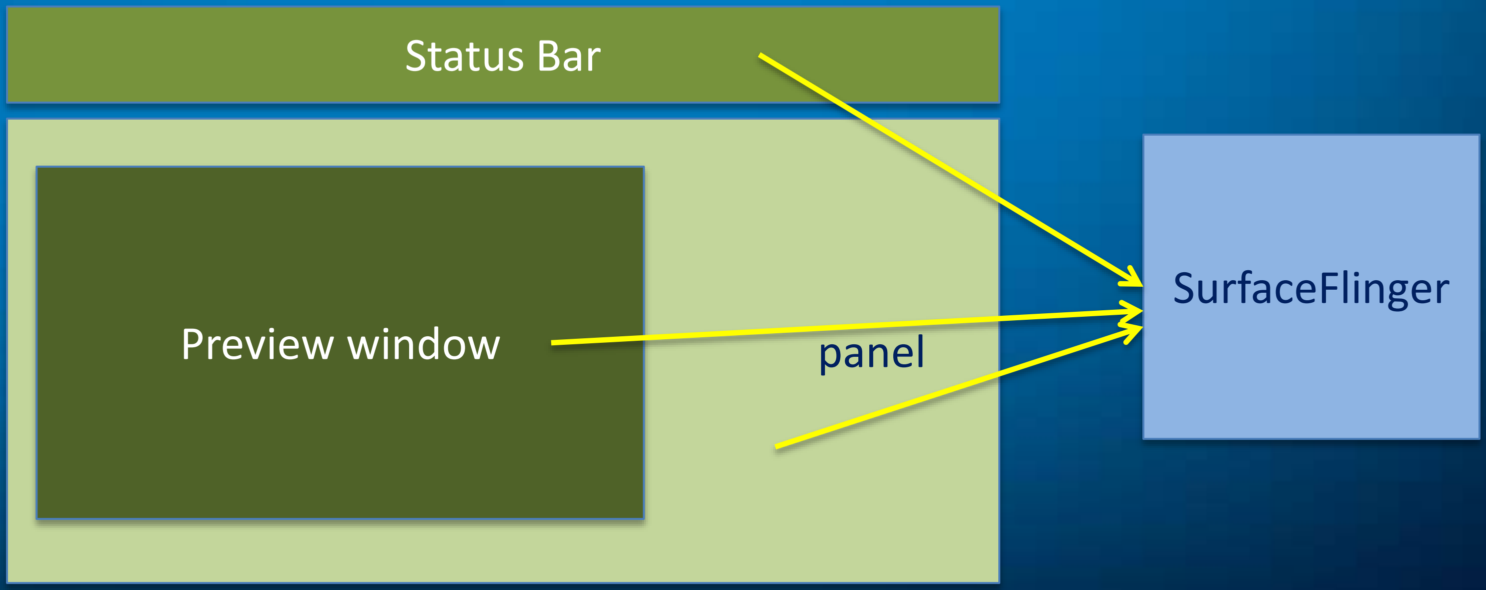


Android-FPS Implementations

- Real-time FPS
  - Show FPS on screen and update in configured frequency
- Post-processing FPS
  - Output metrics of whole term of running into file
- Application FPS
  - Specially designed applications to get app-specific FPS metrics

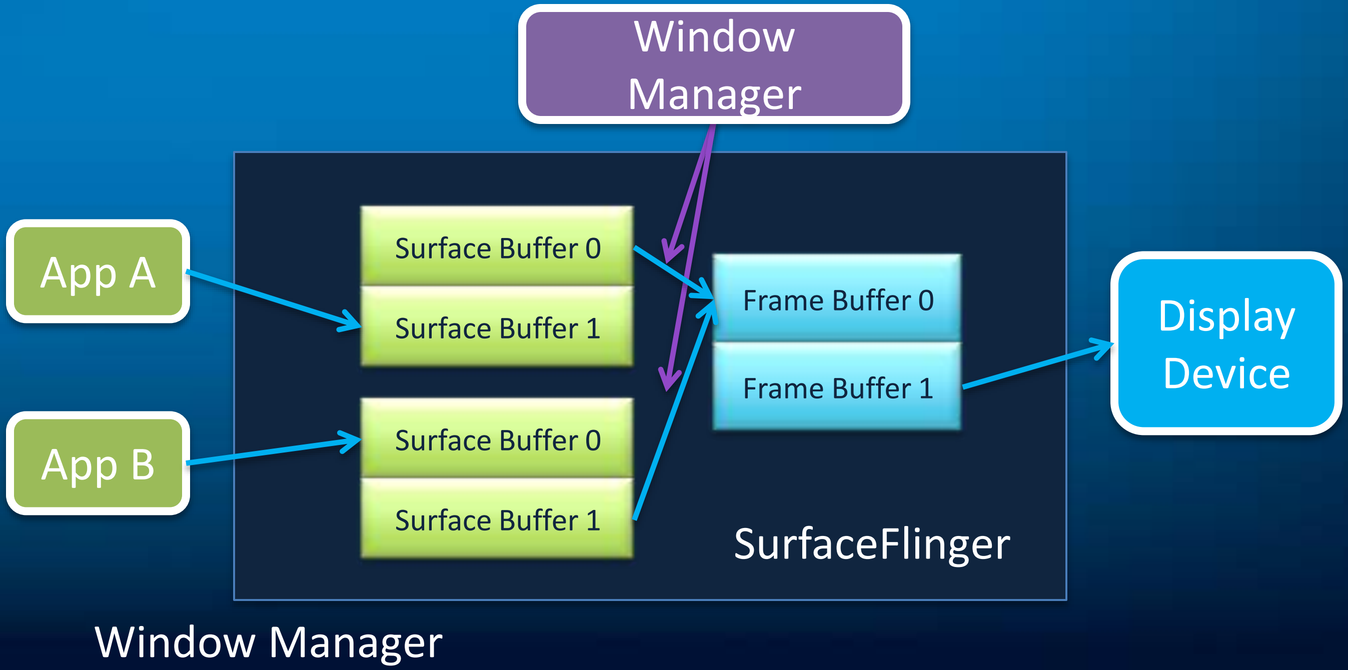
Meter-FPS Example (1): Camera Application

- Different areas may have different FPS values
  - One FPS value is not enough to reflect the application behavior

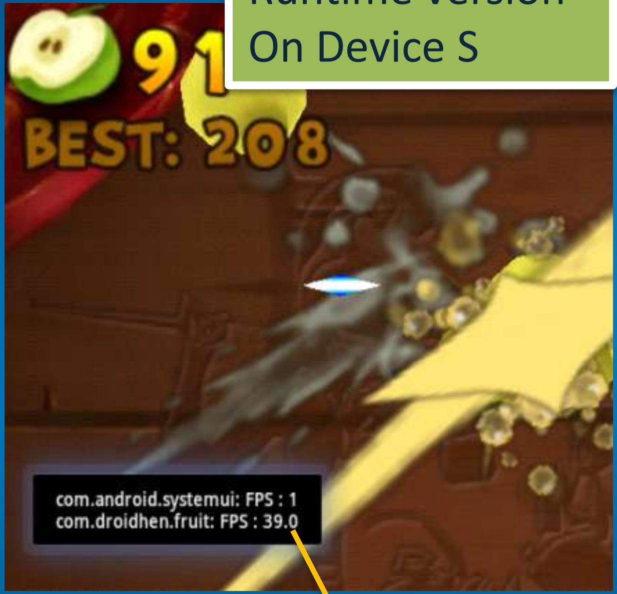


Meter-FPS Example (2): Apps Switching

- The compositing window manager generates the app-switch animation
  - Applications do not draw during apps switching

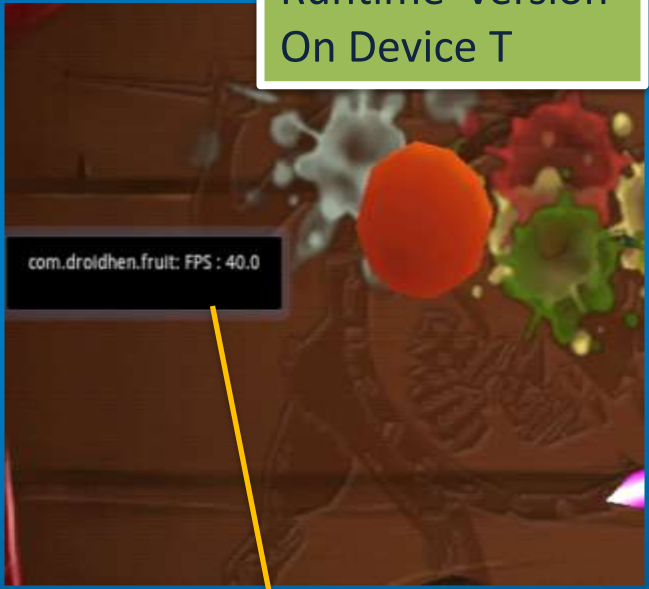


Meter-FPS Example (3): Real-time FPS



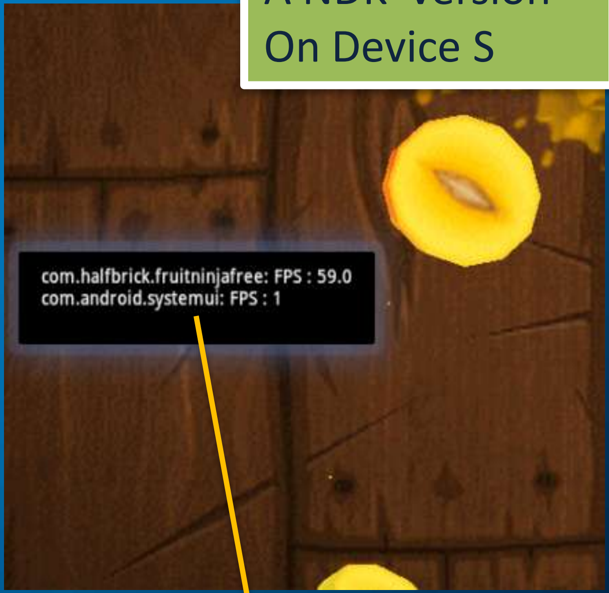
```
com.android.systemui: FPS : 1
com.droidhen.fruit: FPS : 39.0
```

Runtime version
On Device S



```
com.droidhen.fruit: FPS : 40.0
```

Runtime version
On Device T



```
com.halfbrick.fruitninjafree: FPS : 59.0
com.android.systemui: FPS : 1
```

A NDK version
On Device S

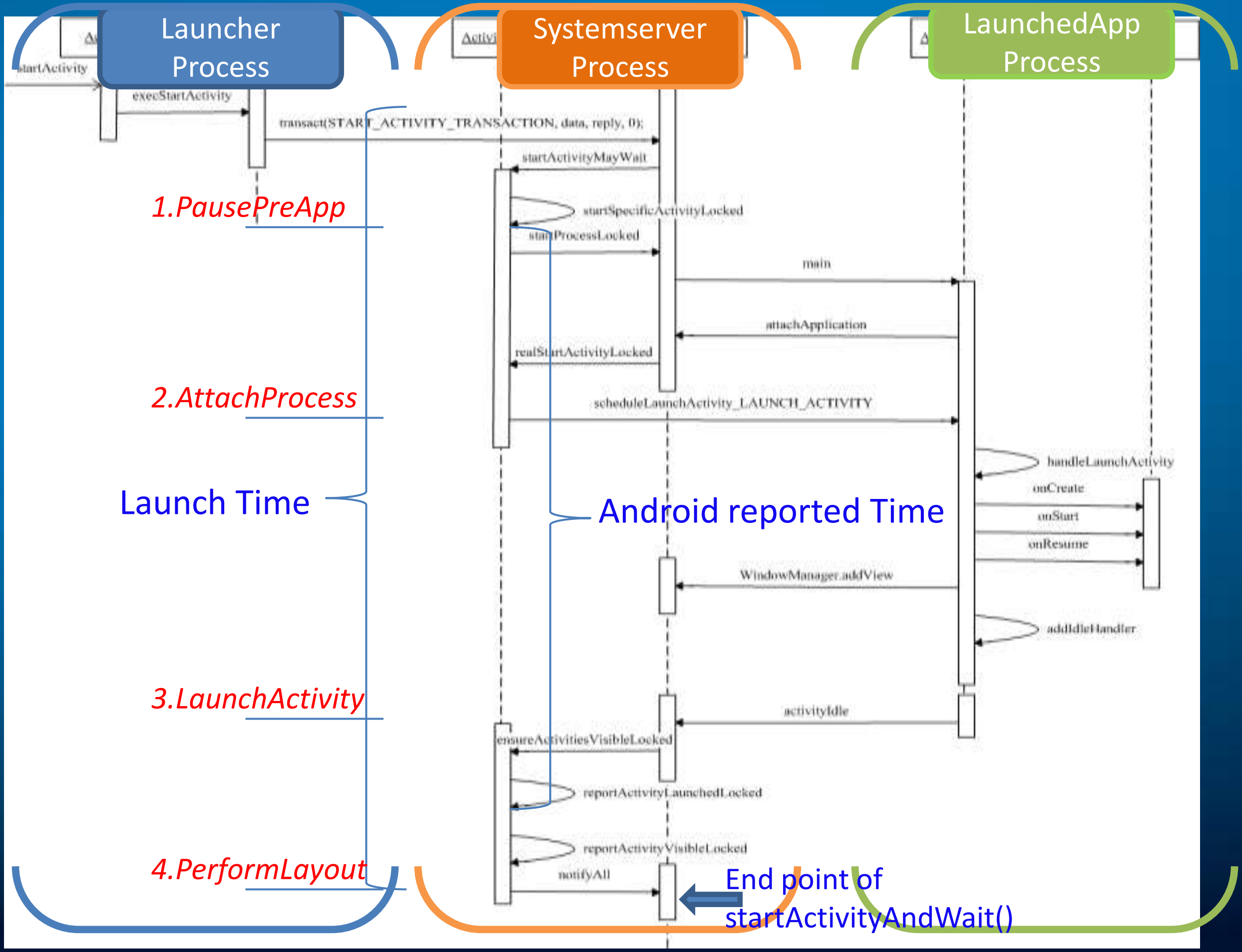
Fruit FPS = 39
systemUI FPS = 1

Fruite FPS = 40

FPS = 59
systemUI FPS = 1

Android Tool for Metrics: App-Launch

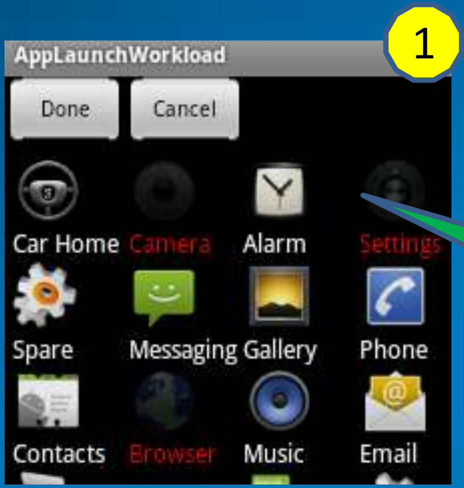
- A tool to characterize application's launch time



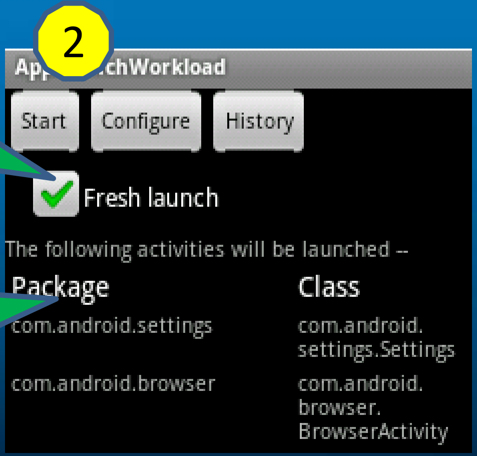
3

Android App-Launch Usage

- AppLaunchWorkload.apk
  - Install->Configure ->Start



Configure: select or deselect applications

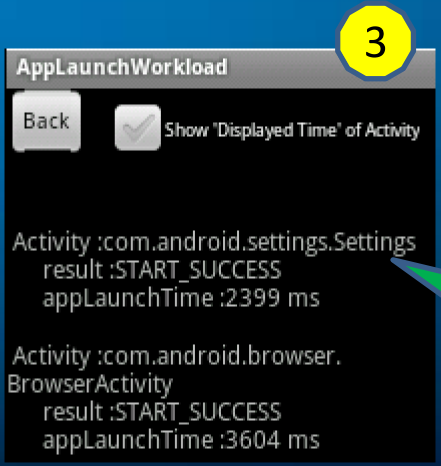


The following activities will be launched --

Package	Class
com.android.settings	com.android.settings.Settings
com.android.browser	com.android.browser.BrowserActivity

Fresh or warm launch

Selected applications

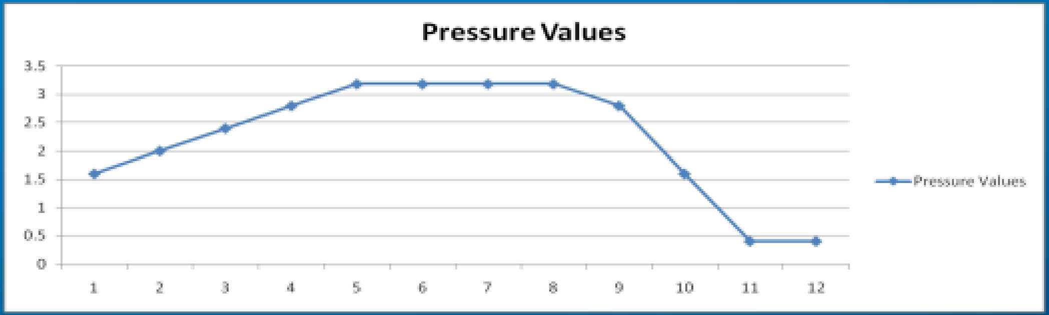


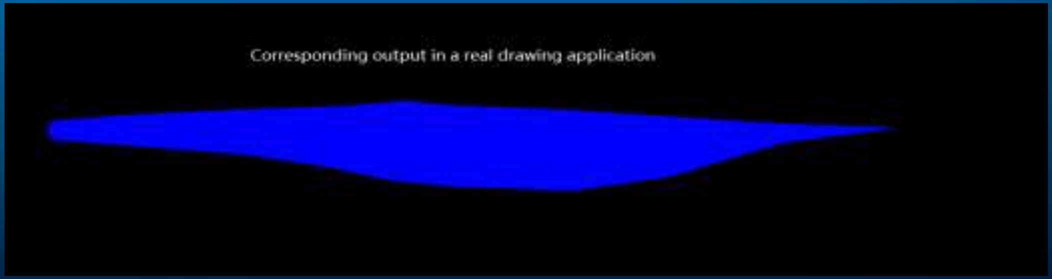
```
Activity :com.android.settings.Settings
    result :START_SUCCESS
    appLaunchTime :2399 ms

Activity :com.android.browser.
BrowserActivity
    result :START_SUCCESS
    appLaunchTime :3604 ms
```

Outputs: result status and data

Android Tool for Metrics: Touch-Pressure

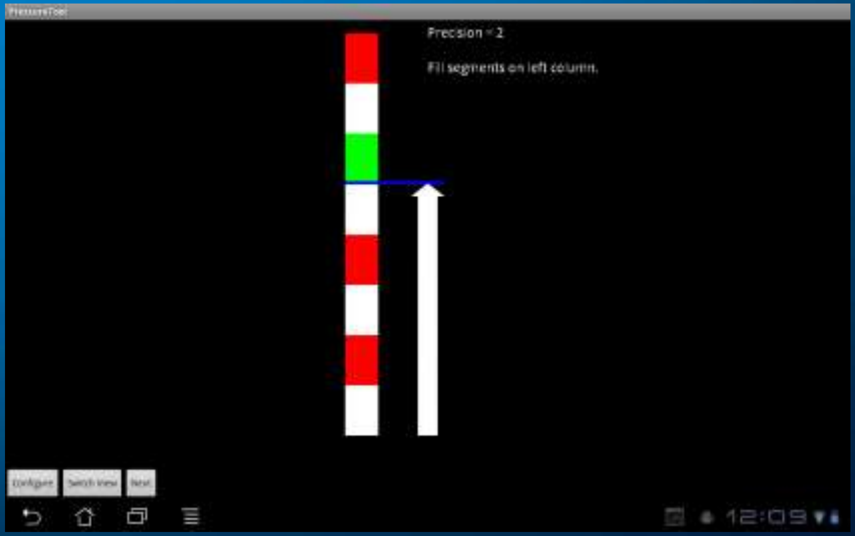
- A tool to get the touch pressure value
  - Pressure is used extensively as natural control
    - Drawing, playing music instruments, gaming, etc.
- A Press in a real device





Touch Pressure Resolution Measurement

- Touch Pressure Resolution
  - – # different pressure values supported by the system
  - – Higher resolution means finer pressure control
- The tool is designed as a game
  - Press the screen to fill in the segments
  - Reflect the real control precision



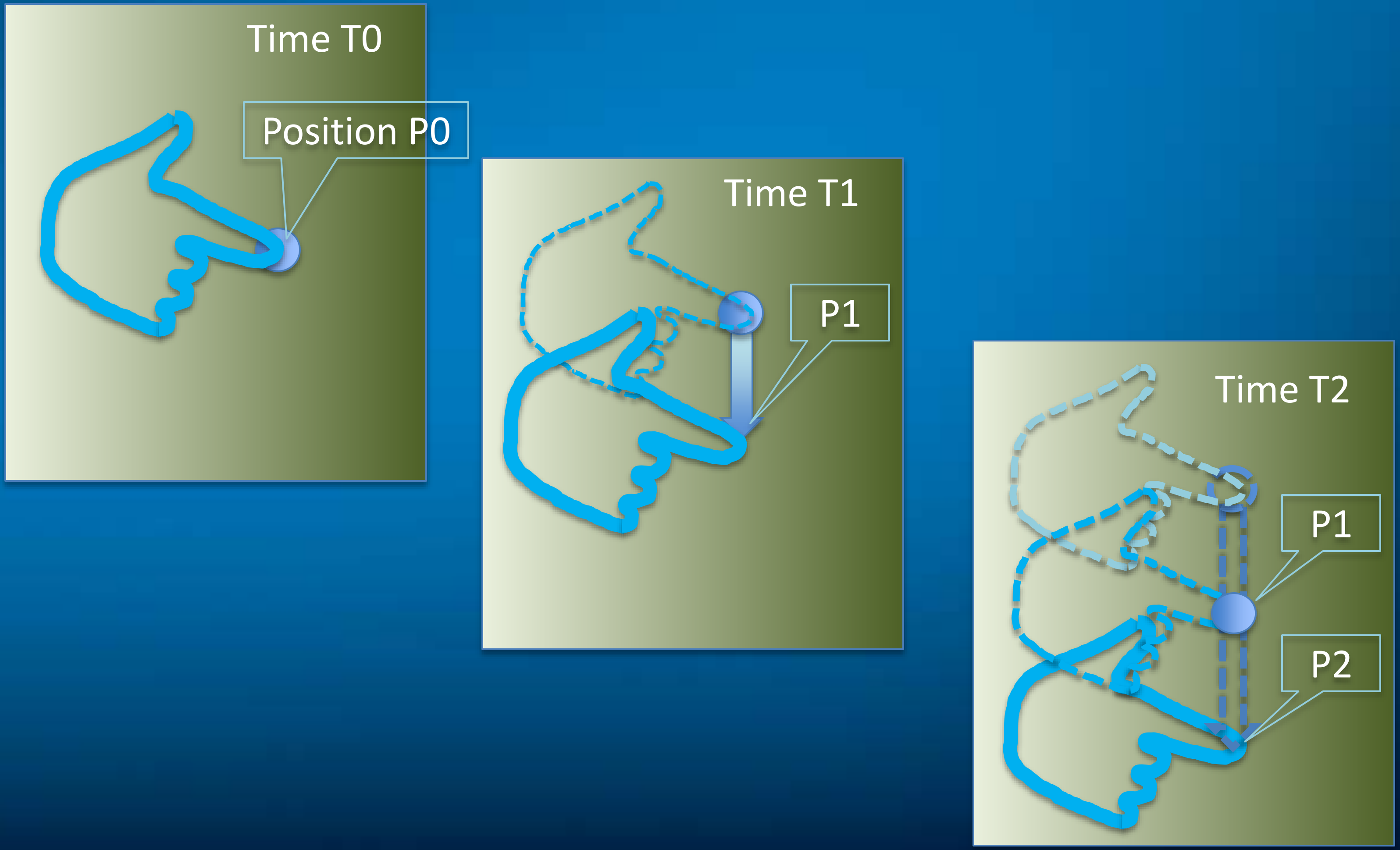
Agenda

- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

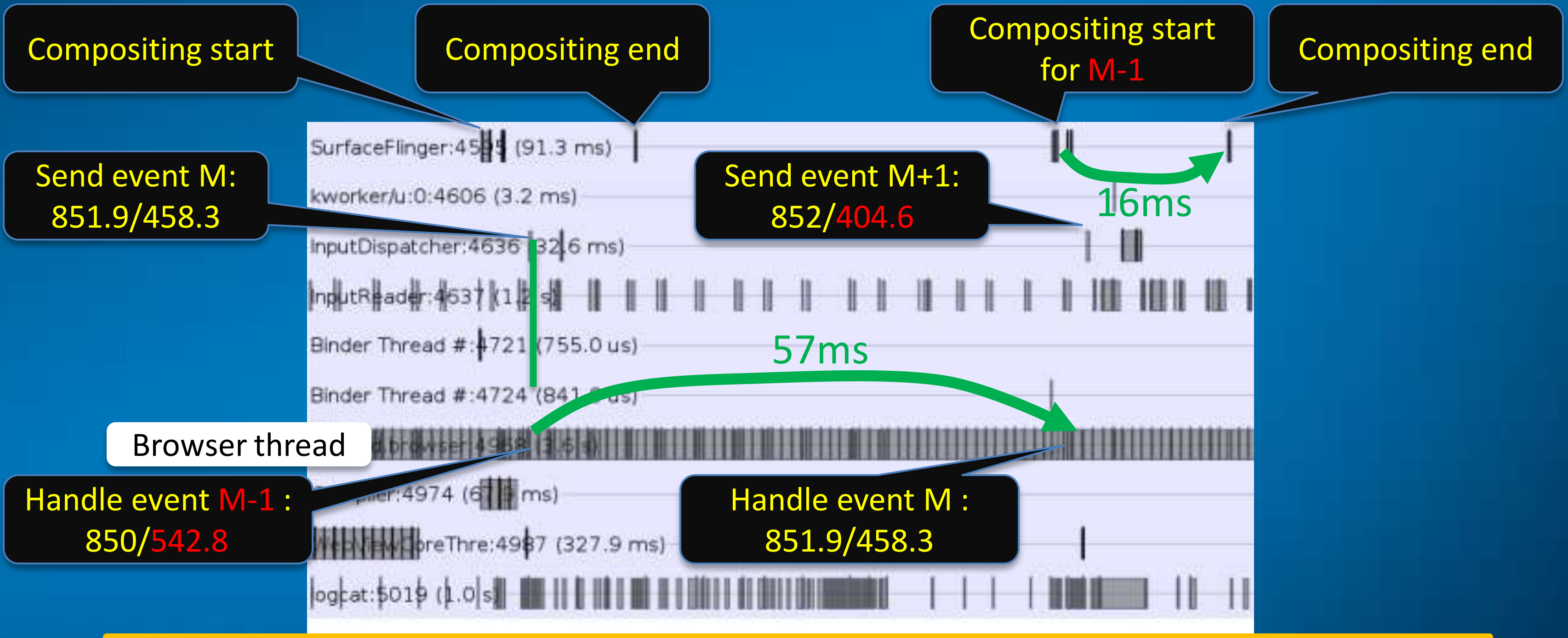
Case Studies with UXtune Toolkit

- Analysis of browser scroll lag distance
- Analysis of FPS bottleneck in MOTO racing game

Browser Scroll Lag Distance



UXtune Analysis of Lag Distance



Lag distance in vertical = 542.8 – 404.6 pixels

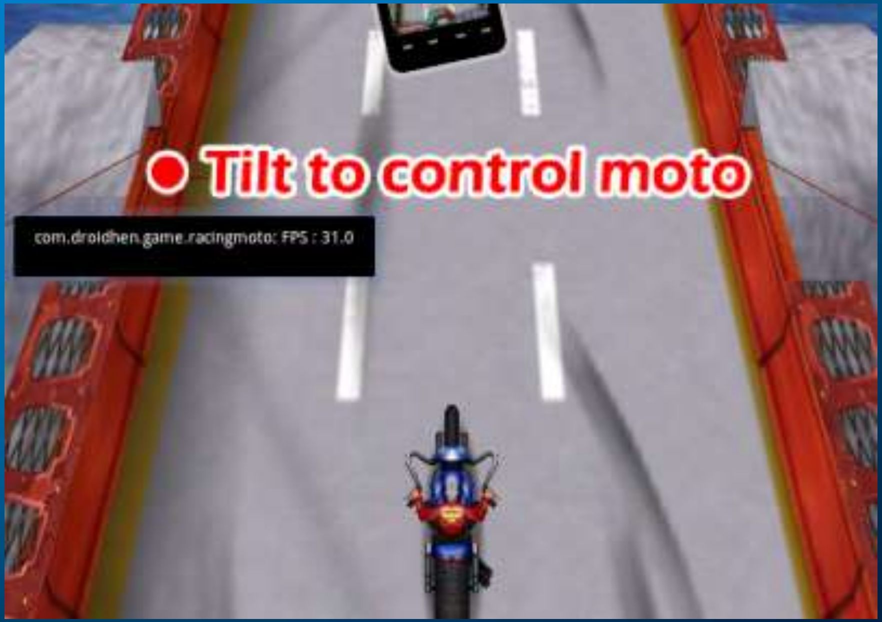
- Poor drawing performance causes long lag
- UI thread wastes time on outdated events

Case Studies with UXtune Toolkit

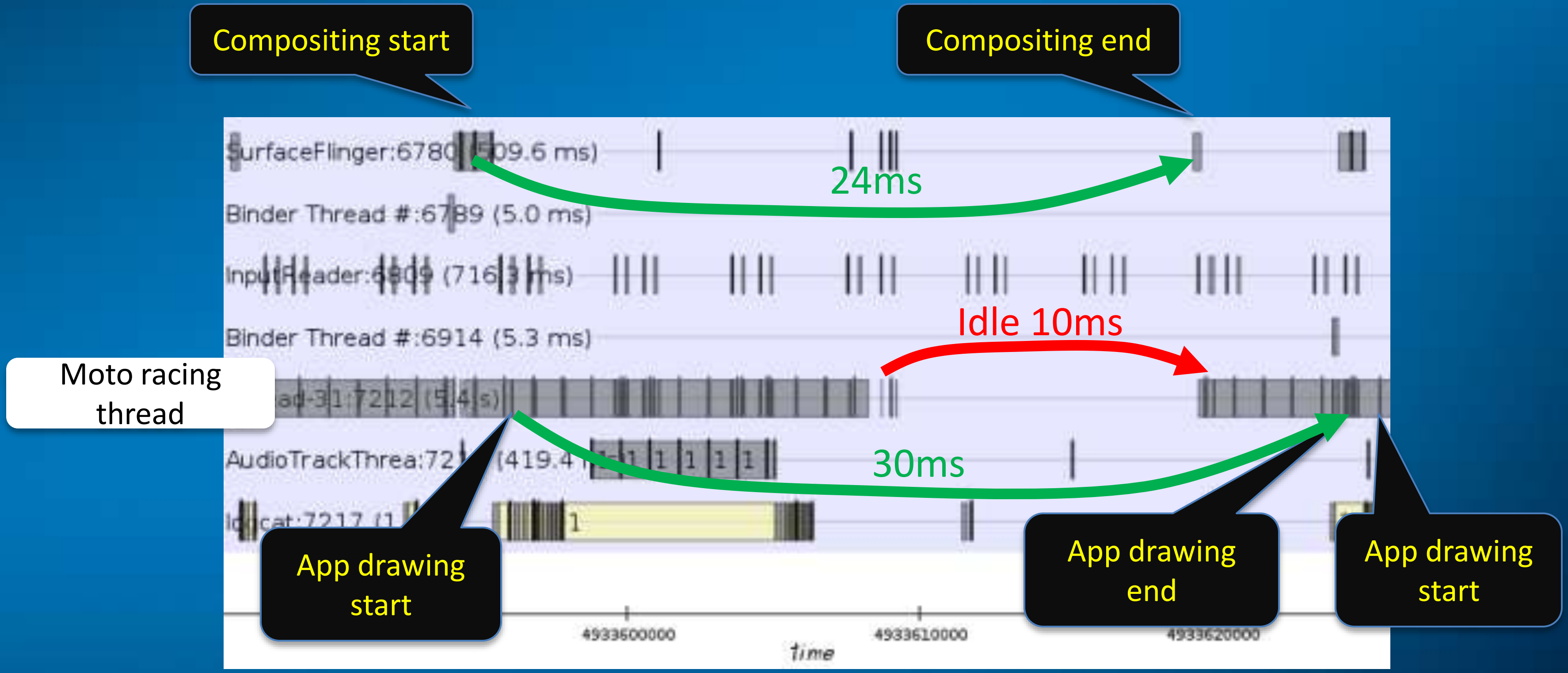
- Analysis of browser scroll lag distance
- Analysis of FPS bottleneck in MOTO racing game

Racing Game Introduction

- MOTO racing game is popular in Android market



UXtune Analysis of MOTO Racing



- Idle time in both app and drawing threads
- The root cause has been identified

Agenda

- Optimization methodology and toolkit
- The inputs: Input-Gestures
- The process analysis: UXtune
- The outputs: meter-FPS, app-launch, touch-pressure
- Case Studies with UXtune toolkit
- Summary

Summary

- UXtune is an engineering toolkit for Android user interaction analysis and optimization
- Tuning user interaction requires to understand the state transitions upon user inputs. We need,
  - Repeatable inputs to operate the device
  - Correlation of events between the analyzed entities
  - Metrics outputs to characterize the state transitions