Delivering Highly Optimized Android* Runtime (ART) and Web Runtime on Intel® Architecture

Haitao Feng – Staff Software Engineer, Intel Corporation
Jonathan Ding – Principal Engineer, Intel Corporation

STTS004

Agenda

- Delivering Highly Optimized Android* Runtime (ART) on Intel® Architecture based devices
  - Intel ART Contributions and Extensions
  - Case Study on Compiler Optimization
  - Case Study on Garbage Collection Optimization
- Delivering Highly Optimized Android Web Runtime on Intel Architecture based devices
  - What's New in latest Android Web Runtime
  - Case Study on Performance Optimization
  - Case Study on Power Optimization
  - Case Study on Memory Optimization

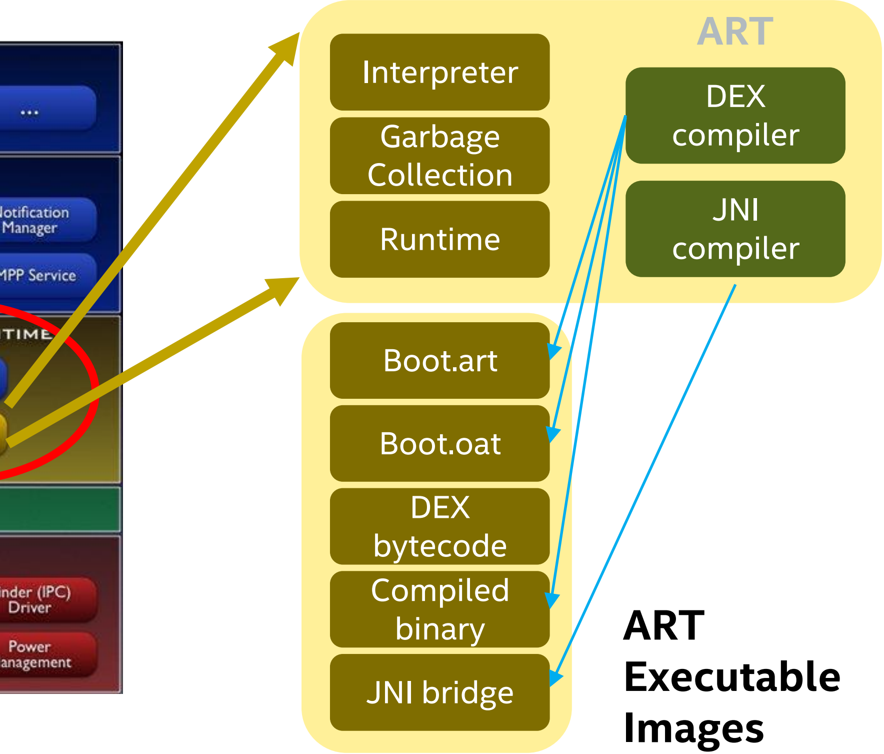
Agenda

- Delivering Highly Optimized Android* Runtime (ART) on Intel® Architecture based devices
  - Intel ART Contributions and Extensions
  - Case Study on Compiler Optimization
  - Case Study on Garbage Collection Optimization
- Delivering Highly Optimized Android Web Runtime on Intel Architecture based devices
  - What's New in latest Android Web Runtime
  - Case Study on Performance Optimization
  - Case Study on Power Optimization
  - Case Study on Memory Optimization

Android* and ART (Android Runtime)

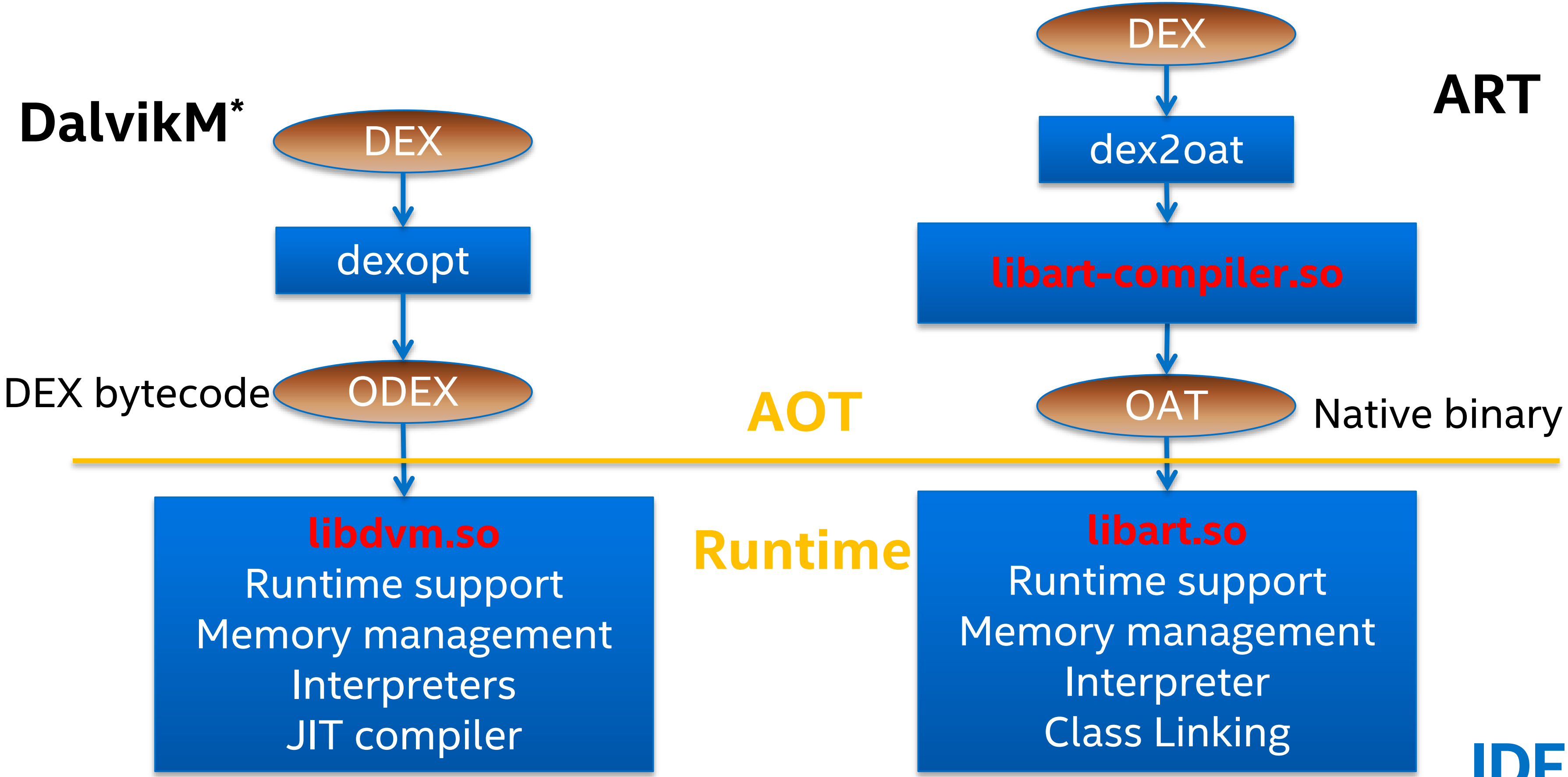
Pre-Lollipop Release:

Lollipop Release:



IDF15

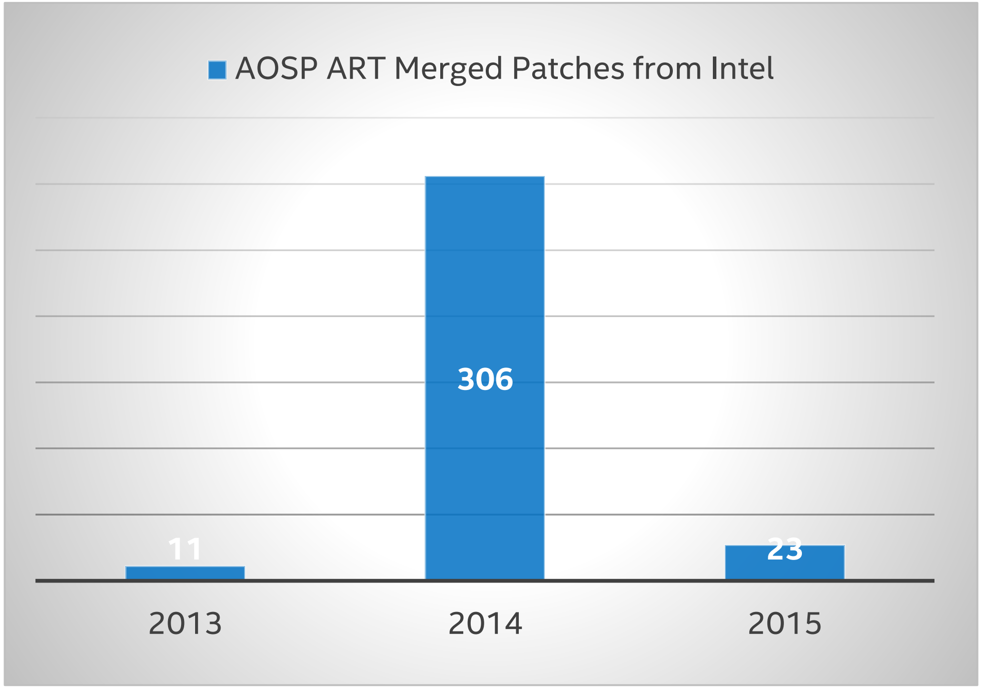
Dalvik* VM vs. Android* Runtime DEX Execution Flow



Intel AOSP Android* Runtime Contributions

Contribution Distribution

- 64-bit enabling and optimizations
- 32-bit code generator optimizations
- Runtime/GC Optimizations
- Debugging enhancement
- Bug fixings



IDF15

Intel Android* Runtime Extensions

An extensible plugin library (libart-extension.so) based on AOSP, which adds additional compiler optimizations:

- Extensive Method Inlining
- Invariant Hoisting/Sinking
- Non-Temporal Move
- Loop Tiling
- Loop Unrolling
- Vectorization

Inside in Android* Products on Intel® platforms

Extensive Method Inliner

- Method Inlining, which replaces a function call site with the body of the callee, is a key optimizations for JVMs
- However, AOSP Method Inliner could only handle very small methods:
  - Empty methods
  - Return of constant
  - Instance getter
  - Return of parameter
  - Instance setter
- Intel's Method Inliner was architected for extensibility and used generic framework to allow inlining of any method body
- Benefits:
  - Effectively turned some methods into leaves
  - Inlining in loops exposed many opportunities

Case Study – Method Inlining and Loop Optimizations

```
int f(int i) {
  return i + 1;
}
```

Before

```
for (i = 0; i < N; i++)
{
  tab[i] = f(i);
}
```

Common Compiler Code

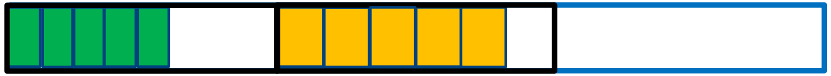
- Method Inlining
- Loop Versioning
- Null Pointer Check Elimination
- Array Bound Check Elimination
- Non Temporal Move

After

```
if (tab != nullptr && tab.length < N) {
  for (i = 0; i < N; i++) {
    // no need to check null pointer
    // no need to check array bound
    tab[i] = i + 1;  // non temporal
  }
} else {
  for (i = 0; i < N; i++) {
    tab[i] = i + 1;
  }
}
```

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - Object allocation and garbage collection:



Case Study - Homogenous Space Compaction

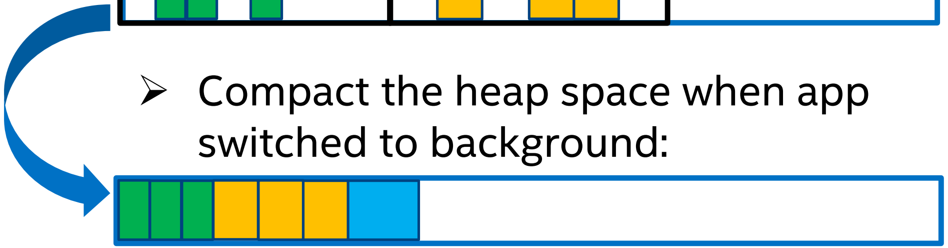
- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

IDF15

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

  - ➢ Compact the heap space when app switched to background:



  - ➢ Re-construct the heap space when app switched to foreground:

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

  - ➢ Compact the heap space when app switched to background:

  - ➢ Re-construct the heap space when app switched to foreground:

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

  - ➢ Compact the heap space when app switched to background:

  - ➢ Re-construct the heap space when app switched to foreground: Pause!

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

  - ➢ Compact the heap space when app switched to background:

  - ➢ Re-construct the heap space when app switched to foreground: Pause!

- Homogenous Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

Case Study - Homogenous Space Compaction

- Previous Heap Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

  - ➢ Compact the heap space when app switched to background:

  - ➢ Re-construct the heap space when app switched to foreground: Pause!

- Homogenous Space Compaction
  - ➢ Object allocation and garbage collection: Fragmentations!

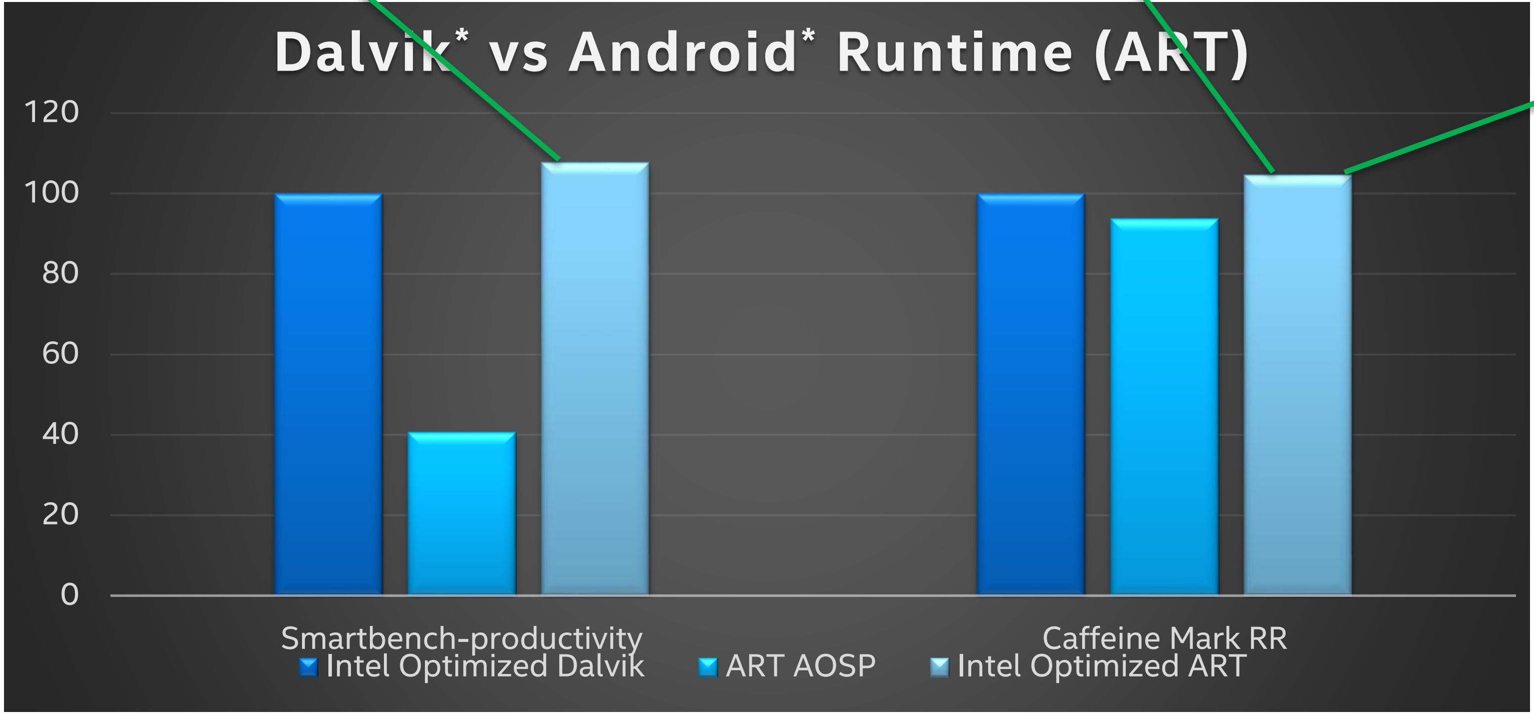
  - ➢ Compact the heap space when app switched to background:

  - ➢ No pause as we do not need to re-construct the heap space, this improves the responsiveness.

Performance Data with Intel Optimized Android* Runtime

Method Inlining
Tiling
Invariant Hoisting/Sinking

Select Optimization
Full Unroller
CSE



Normalized to Dalvik: Above 100 means ART is better

IDF15

Agenda

- Delivering Highly Optimized Android* Runtime (ART) on Intel® Architecture based devices
  - Intel ART Contributions and Extensions
  - Case Study on Compiler Optimization
  - Case Study on Garbage Collection Optimization
- Delivering Highly Optimized Android Web Runtime on Intel Architecture based devices
  - What's New in latest Android Web Runtime
  - Case Study on Performance Optimization
  - Case Study on Power Optimization
  - Case Study on Memory Optimization

Emerging Technologies and Innovations in Web

App (HTML/JS/CSS)

- WebRTC
- asm.js
  - Towards speed of native
  - Emscripten, etc.
- web components
  - Create your own HTML tag
  - polymer etc.

```
<my-tag content="hello world"></my-tag>
```

Web Runtime	Hardware & OS

- 64-bit Android*
- Chromium* WebView

	WebView v30	WebView v33	WebView v36
WebGL	×	×	√
WebRTC	×	×	√
WebAudio	×	×	√

source: Google (link)

Intel Embraces and Optimizes them for Better User Experience

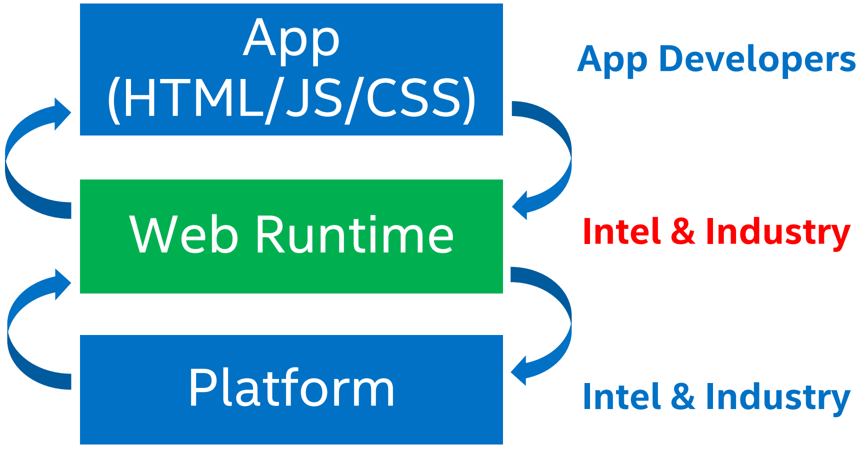
IDF15

User Experience of HTML5 Applications

- User Experience is correlated to Performance, Power Efficiency, and Resource Utilizations

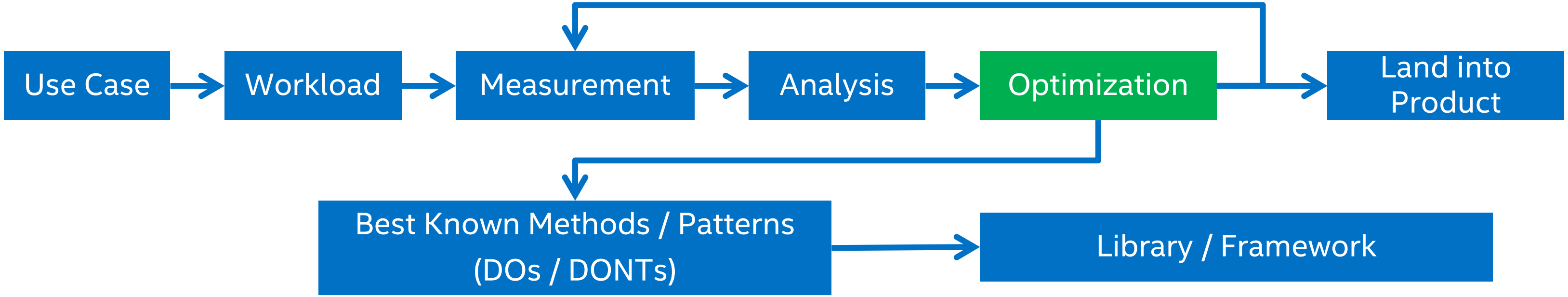
User Experience	Correlated To
Response Time	Performance
Smooth Animation	Performance
Battery Life	Power Efficiency
Navigating among multiple web sites	Memory Footprint
...	...

- Optimization Opportunities comes from Entire Stack



Methodology of Optimization

- Shared Wisdom in optimizing both App and Web Runtime



<u>Use Case</u>

```
var x, y;
// what is the
actual type of
x and y?
```

<u>Analysis</u>

Type in JavaScript*

- Flexible for Developers
- Challenge for Performance

<u>Optimizations</u>

- Raw Idiom for Developers

  ```
  var x = x | 0;
  ```

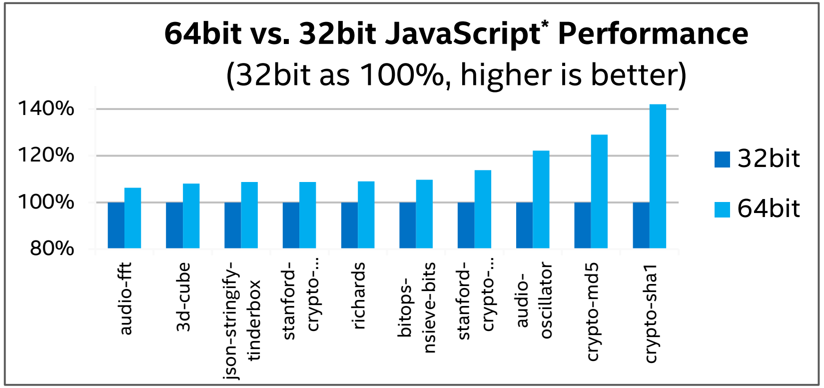
- Idioms → Library (& tools) for Developers
  - asm.js + Emscripten
- New APIs by Runtime for Developers
  - TypedArray: `var array = new Int32Array(buffer)`
- Runtime (transparent for Developers)
  - JIT + Type Speculation in V8

Case Study – Performance Optimization

- → 64bit
  - Performance boost due to more registers in wider length etc.
  - Web Runtime to assist the transition, e.g., sign extension

32bit	-1			FFFF	FFFF
64bit					
2^{32}-1		0000	0000	FFFF	FFFF
	-1	FFFF	FFFF	FFFF	FFFF

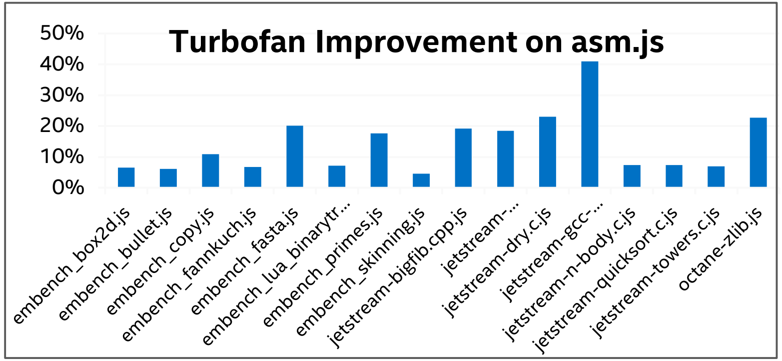
optimization



source: Intel, measured on Intel® N2820 @2.13GHz with Ubuntu 13.10 and V8 r26534

- No need for positive
- A number only need extend once → Merge

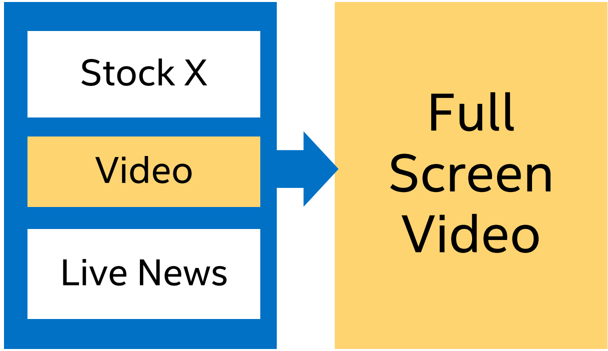
- → TurboFan of V8
  - Powerful IR (Sea of Nodes)
  - First Milestone targeting asm.js
  - Intel optimizations in codegen
- → SIMD.js etc. (Session SFTS004)



source: Intel, measured on Intel® N2820 @2.13GHz with Ubuntu 13.10 and V8 (x64) r26534

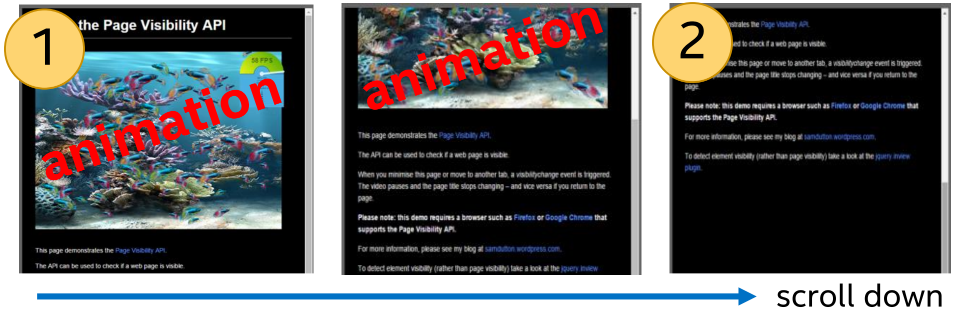
Case Study – Power Optimization

- Do Less – Example of Video



- [Optimization] Put Video in an Overlay to directly composite by HWC, rather than go to GLES first
- [Optimization] If Full Screen, No need to render underlying content
- About 30%† power saving in total

- Do Less – Example of Web Surfing
  - Page Visibility 2 (at iframe level)



  - [Optimization]
    - Runtime tracks this and notifies visibility change
    - App (code by developers) watches it and stops animation accordingly
  - About 60%† power saving in prototype
  - Discussion in W3C

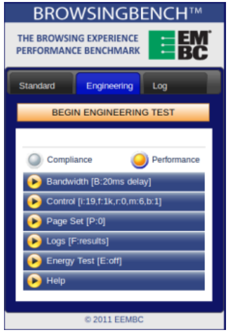
IDF15

† source: Intel, measured on Intel® Atom™ with Android* 4.4 and Chrome* M40

Case Study – Memory Optimization

- Memory Footprint – important for web surfing user experience
  - Chrome*: Normally, one tab → One Process
  - Avoid Memory Bloat: Android* LMK (Low Memory Killer) may pick and kill process under memory pressure → Re-spawn & load again when revisit page

Use Case
(10 pages x load 7 times)



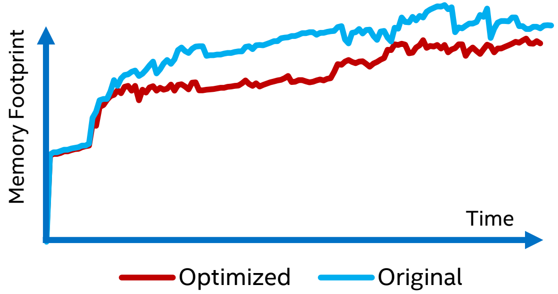
screenshot of EEMBC* browsingbench*

Data, Analysis and Optimization

- Cache ⇔ Performance Tradeoff
- Opportunities of Better Purge Strategy
  - Data of GIF Decoding
  - Data not referenced by any web page
  - JavaScript* GC
  - Shared Memory between processes

Result: Similar Perf. & >10% less footprint

Memory Footprint of Page Loading
(Lower is Better)



source: Intel, measured on Intel® Atom™ with Android 4.4 and Chrome M40

IDF15

Summary & Next Steps

- For the Android* Lollipop release, Intel put enormous efforts to make sure both Android Runtime (ART) and Web Runtime are highly optimized on Intel® Architecture based devices
- Intel optimized Android Runtime with a better AOT compiler and enhanced GC and will work with partners to optimize it further
- Web Apps' User Experience is improved by optimizations from both and runtime that Intel and industry is continuously enhancing
- For Application Developers, know and practice more about:
  - 64bit, asm.js, SIMD.js etc. for better performance
  - Hardware Acceleration, Page Visibility, etc. for power efficiency
  - Cache and GC strategy etc. to satisfy the budget of resource like memory

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

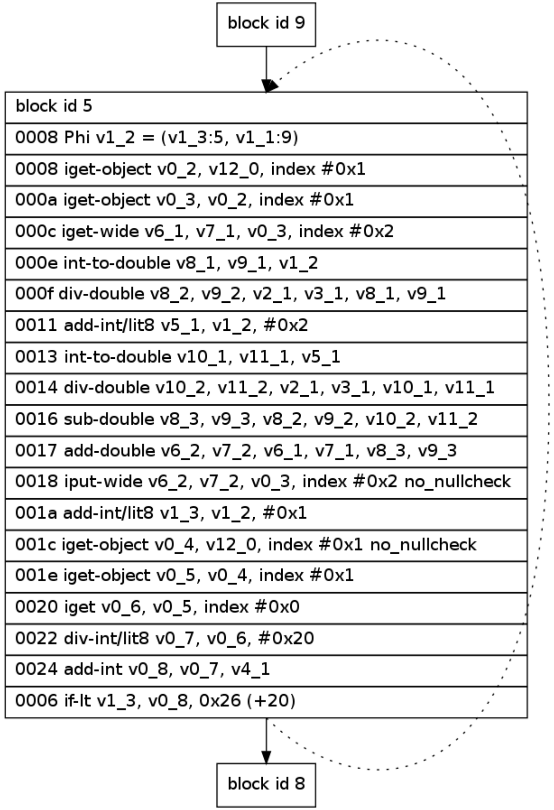
Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as "anticipates," "expects," "intends," "plans," "believes," "seeks," "estimates," "may," "will," "should" and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel's actual results, and variances from Intel's current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be important factors that could cause actual results to differ materially from the company's expectations. Demand for Intel's products is highly variable and could differ from expectations due to factors including changes in the business and economic conditions; consumer confidence or income levels; customer acceptance of Intel's and competitors' products; competitive and pricing pressures, including actions taken by competitors; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel's gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; and product manufacturing quality/yields. Variations in gross margin may also be caused by the timing of Intel product introductions and related expenses, including marketing expenses, and Intel's ability to respond quickly to technological developments and to introduce new features into existing products, which may result in restructuring and asset impairment charges. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Results may also be affected by the formal or informal imposition by countries of new or revised export and/or import and doing-business regulations, which could be changed without prior notice. Intel operates in highly competitive industries and its operations have high costs that are either fixed or difficult to reduce in the short term. The amount, timing and execution of Intel's stock repurchase program and dividend program could be affected by changes in Intel's priorities for the use of cash, such as operational spending, capital spending, acquisitions, and as a result of changes to Intel's cash flows and changes in tax laws. Product defects or errata (deviations from published specifications) may adversely impact our expenses, revenues and reputation. Intel's results could be affected by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel's ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. Intel's results may be affected by the timing of closing of acquisitions, divestitures and other significant transactions. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the company's most recent reports on Form 10-Q, Form 10-K and earnings release.

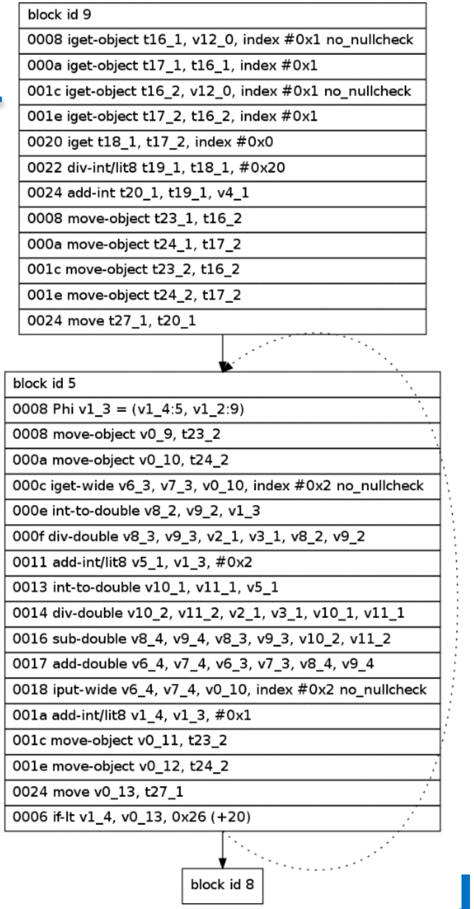
Backup

Case Study on Compiler Optimization Part 1



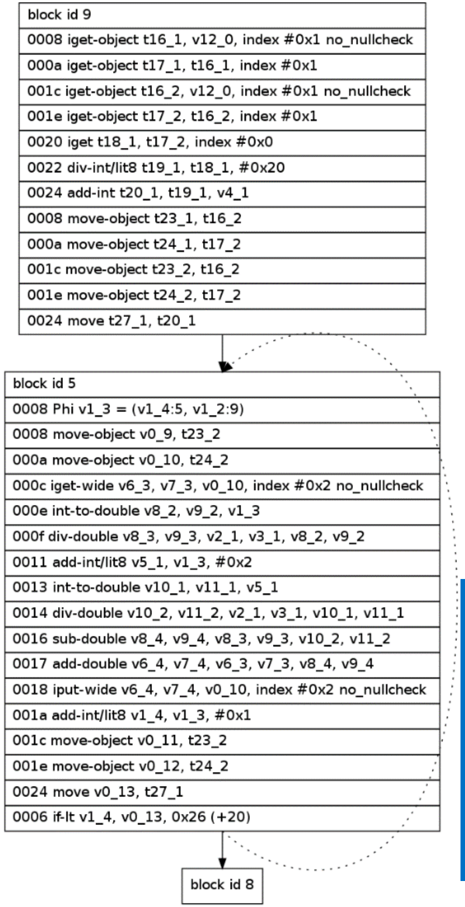
Hoisted the invariants
No RA so there are a few useless moves here

Move objects in loop to respect GC requirements



IDF15

Case Study on Compiler Optimization



Invariants Hoisted

Tiling:

- Provides a means to remove the suspend check
- This enables loop optimizations for the loop

Registerization Done

Tiled Loop