Pretoria

Tantalum

Elop -> null

Clojure Android?

Scala Android?

C# Android?

JS Android?

TypeScript Android?

C? D? F#?

Java!

Java!

RX?

Java!

~~RX?~~

Java!
Functional
Reactive
Lambdas
Async
Transparent
Performanc
e

-> All new, better

Performance Psychology:
To increase developer performance..

- Decrease Code Errors
  - Fail fast
  - Explicit threading at object creation, not invocation
  - Pure function trees
    - Only leaf nodes have side effects
- Decrease Tangential Work
  - Minimize boilerplate
  - Provide in-flow system testing
  - Focus on algorithm, not plumbing
    - Concurrent reactive data structures
- Increase Code Insight
  - Distributed algorithm transparency
  - Explicit pre-conditions
  - Show and annotate the origin in _your_ code of any messages
  - Flat lambda-passing for less hierarchy

Performance Information Logistics: To increase runtime performance..

- Decrease Work In Progress (WIP)
  - WIP = Partially finished goods
  - Depth-first task tree traversal
  - Path-independent optimization of convergence on eventually consistent end state
- Increase resource utilization rate
  - Eliminate blocking using explicit dependency chains
  - Tuned thread pools for
    - Common work backlog
    - Optimal concurrency
    - Minimum peak memory load

Here and Now NEW TOPIC

Search things happening now

Open Invite: Kista 3D Pizza

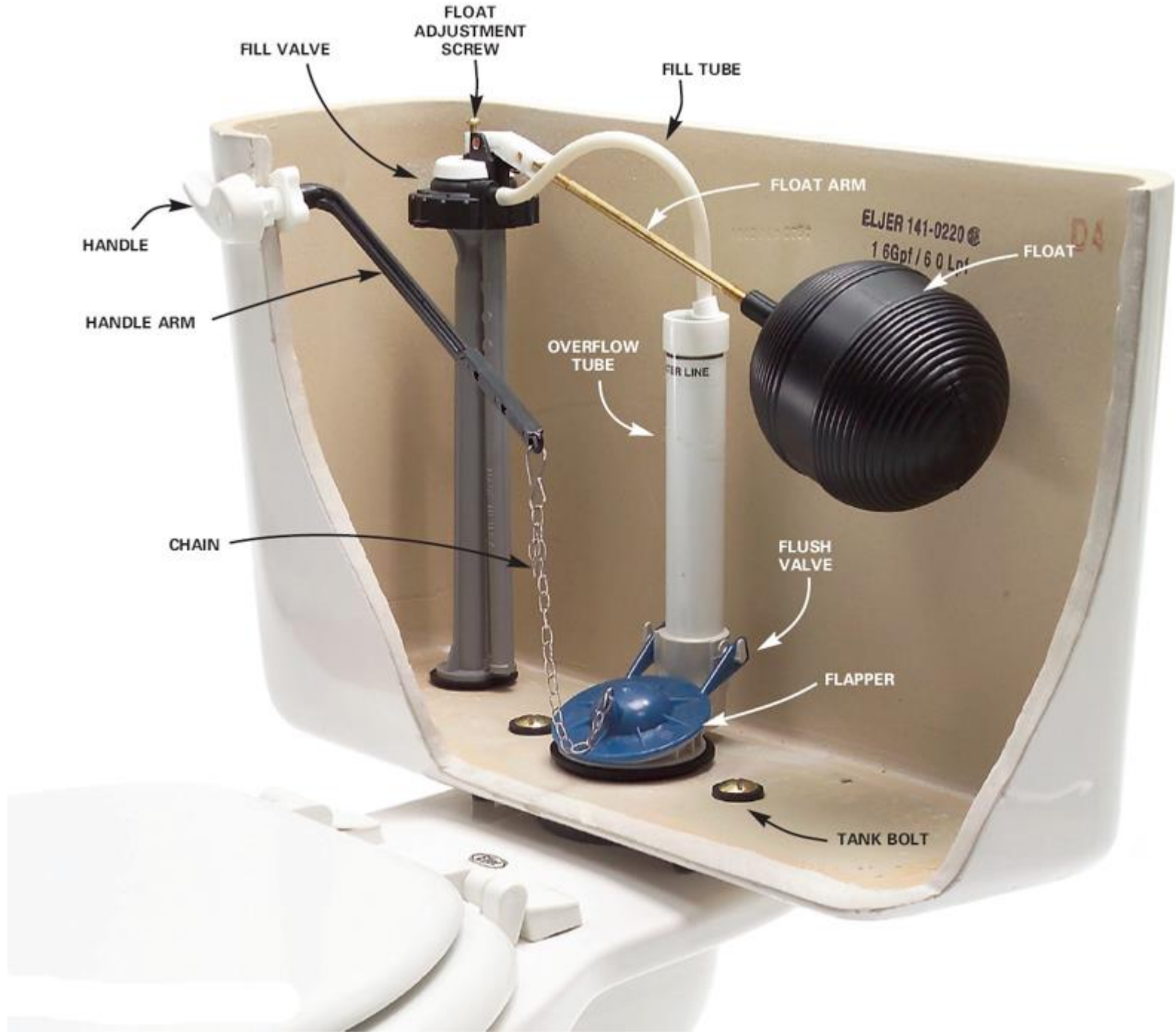
Here are some nice places to go for lunch

Here and Now NEW TOPIC

Search things happening now

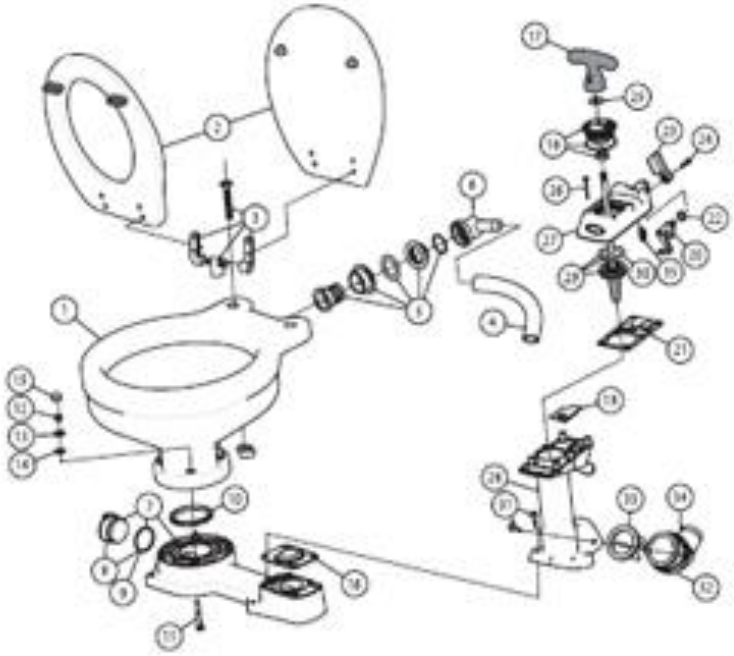
10% discount on headbands at the ski shop 200m north from you with this coupon.

Open Invite: Kista 3D Pizza


FLOAT
ADJUSTMENT
SCREW
FILL VALVE
FILL TUBE
FLOAT ARM
HANDLE
HANDLE ARM
FLOAT
OVERFLOW
TUBE
CHAIN
FLUSH
VALVE
FLAPPER
TANK BOLT

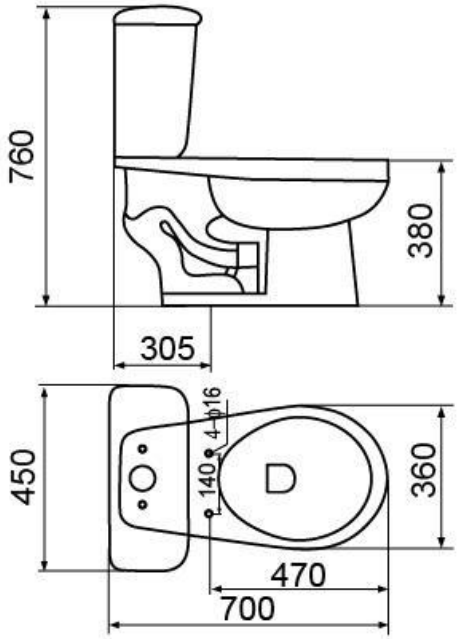
Kit 1

- Connect plumbing for each use
- Disconnect and clean after each use
- Everyone uses it



Kit 2

- Assembled: one step install
- Self-cleaning
- Experimental, open to improvement



What you really want: zero config

Kit 3 “works like a train toilet”

Are you driven to make rugged and reliable machines?

.. or Cargo Cult Development

.. or Cargo Cult Development

“You cannot make a man by standing a sheep on its hind legs. But by standing a flock of sheep in that position you can make a crowd of men.”

-Max Beerbohm

“You cannot make a man by standing a sheep on its hind legs. But by standing a flock of sheep in that position you can make a crowd of men.”

-Max Beerbohm

-> Typical open source library that throws useless stack dumps at you

Reactive Cascade

Q: Ask Why

A: Quickly develop reliable, high performance Android apps

The driver: Great UX

- (skip interface designer blah blah here)

- Developer: **architecture details ->**
  - **Speed**
  - **Trust contract with the user**
  - **Implicitly non-blocking**
  - **Deeply event driven**

The driver: Great UX

- Developer: **architecture details ->**

  Speed

  - start time, close time, persistent initial display state

  - subjective speed (UI) prioritization, bend time and space, throttle peak contention

  - after the visible: high core throughput decreases user-perceptible delays, power use

  Trust contract with the user

  Implicitly non-blocking

  Deeply event driven

The driver: Great UX

- Developer: **architecture details ->**

  Speed

  Trust contract with the user

    - contract: demons in your phone accept a task. And always do it
    - orderly behavior when things go wrong

  Implicitly non-blocking

  Deeply event driven

The driver: Great UX

- Developer: **architecture details ->**

  Speed

  Trust contract with the user

  Implicitly non-blocking

  - There is a correct thread for everything, and no more

    how are you bound? CPU, flash read, flash write, network (dynamic concurrency by available bandwidth)

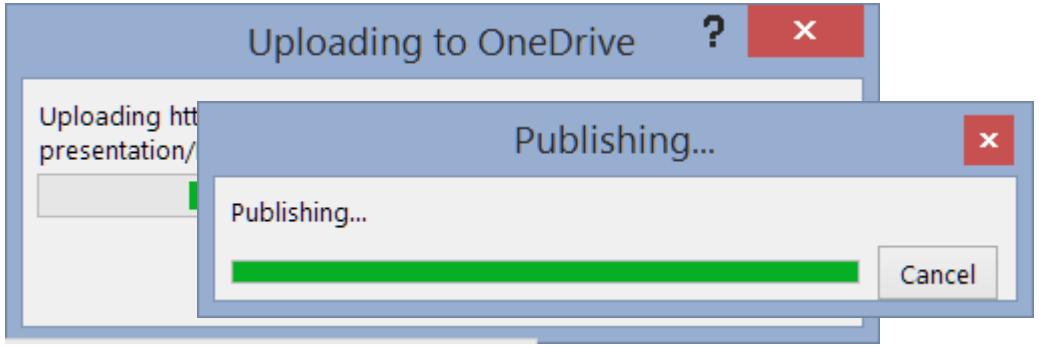
  - Minimal resource usage (concurrent memory use window, concurrent peak, write behind of needed)

  Deeply event driven

The driver: Great UX

- Developer: **architecture details ->**
  - **Speed**
  - **Trust contract with the user**
  - **Implicitly non-blocking**
  - **Deeply event driven**
    - minimal timer loops, throttling and back pressure
    - do things I can see first, the reset whenever (async)

Great UX?



- Publish PDF ->
  - Create PDF is fast
  - But you can't do anything until the net upload finishes (slowly)
  - It then opens a _new_ copy of a webpage (every time, no choice)
    - That forwards
    - That forwards
    - That _downloads_ and shows an web copy of the PDF I published
    - That I do not want. I'm just exporting a PDF to my hard drive


ONLY
POOL


PLATFORM
CLOSED
TEMPORARY
BUS STOP


PLATFORM CLOSED
TEMPORARY BUS STOP
UI
WORKER


NET WRITE
NET READ


FLASH WRITE
FLASH READ

Great Developer Experience (DX)

- Self-validating objects catch and help solve common errors early
- Optimal concurrency is automatic
  - Strong threading model driven by the _creator_ of a object, not the _caller_
- Errors are handled sequentially like other functional and reactive chains
- Zero config and near zero boilerplate
- Your properly threaded algorithm show clearly in one place
- Good defaults are deeply replicable
- Aquarium transparency means you know what is going on
  - Lambdas: all code for a sequence of operations is visible in one place
  - Warnings and errors tell you what you probably need to do
  - Warning and error provide a convenient link to the _most likely cause_ of the error, not just a cryptic result

Cascade is Functional and Reactive

Functional

- Fire once -> immutable
- Strongly typed `.then()` chains
- **Λ**-friendly functional interfaces.then() to compose a chain

  -> immutable value object (VO) after .fork() and successful run

  .cancel() or exception -> immutable
- Example: ImmutableValue<String> s

  s.then(value -> println(value));

Reactive

- Variable changes -> fire sequence
- Compiled synchronous exec
  - unless you change thread group
- `.subscribe()` and `.unsubscribe()`
- Automatic unsubscribe when all subscribers are gc()ed
  - Tail held tree with weak forward reference
- Example: AtomicValue<String> s

  s.subscribe(val -> println(val));

-> Use the right tool for the job -> what makes most sense to you

Functional Example

```
ImmutableValue<Integer> count = new ImmutableValue<>();

// I don’t yet know the value
// I do know what I want to do when the value is determined
count.then(value ->
        println(“The count is “ + value));
..
count.set(34);
// Triggers one time logic run
// Count is now an immutable value object
// Clean for further functional use
```

The point: throw logic around – it can safely happen concurrently on any thread

Functional Example

```
SettableAltFuture<Integer> count = new SettableAltFuture<>(WORKER);
count.then(value ->
   recalculateSheetOne(value); // CORE 1
count.then(value ->
   recalculateSheetTwo(value); // CORE 2
count.then(UI, value ->
         println("The count is " + value)) // CORE 3
..
count.set(34);
```

Atomic operation from any thread triggers transform to immutable value object

Three down-chain actions run concurrently on the UI and two worker threads

Functional Example

```
NET_READ.then(() ->
        return getMessagesServer())
    .then(WORKER, (raw) -> {
        return parseMessages(raw))
    }
    .then(NET_WRITE, (MyDataType parsedList) -> {
        storeToFlash(parsedList);
    }
    .then(UI, (MyDataType parsedList) -> {
        display(result))
    }
    .onError(() ->
        popupError("Stopped at count " + count.get());
```

Reactive Concurrent Example

```
AtomicValue<Integer> incidentAngle = new AtomicValue<>(WORKER, “Item
length”);
count.subscribe(UI, value -> {
    println(“The length is “ + value)       // CORE 1, main thread
}
    .subscribe(degrees -> {
          viewModel.updateAngle(degrees) // CORE 2, concurrent background
}
    .subscribe(value -> {
          return mapValue(value);          // CORE 3, concurrent background
}
    .subscribe(value -> {
          mappedSum::increment);           // AFTER CORE 3, sequential

..
// From any thread
count.set(34);
// Cascade of concurrent downchain actions on each set
```

Reactive Concurrent Example

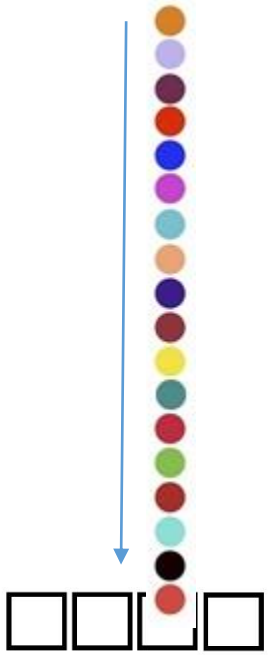
```
AtomicValue<Integer> count = new AtomicValue<>(WORKER);
count.subscribe(value ->
   recalculateSheetOne(value);
count.subscribe(value ->
   recalculateSheetTwo(value);
count.subscribe(UI, value ->
        println("The count is " + value))
..
count.set(34);
```

Atomic operation from any thread triggers state variable change

Three down-chain actions run concurrently on the UI and two worker threads

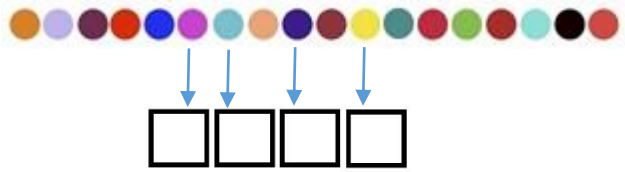
.. every time the state changes

Two Classic Threading Approaches



- Single Thread
  - Logically easy
  - Component steps don't need concurrent design
  - Slow as fuck

- Unlimited or "automatic" thread pool
  - Logically easy, not my concern
  - Unlimited concurrency
  - Context switch and resource use
  - Probably faster, but still slow fuck
    - .. And "out of memory"



Why not use Futures with One Thread?

- A very short story
  - Object A is not yet done
  - Your one thread calls A.get()
  - Deadlock

- Moral: if threads block, you need an unlimited number of threads

Future vs AltFuture

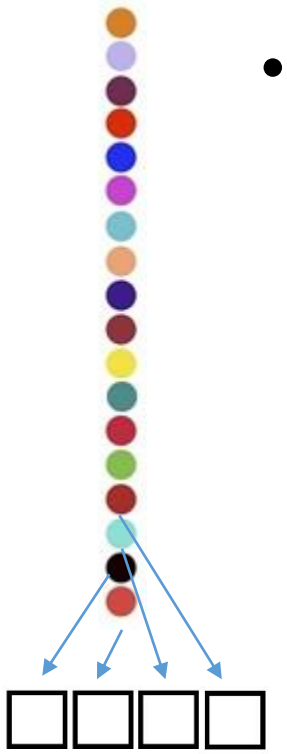
Future

- Runs on any thread
- Blocks any thread at .get()
- Max concurrency not addressed
  - Too few threads -> deadlock
  - Too many -> mem and CPU bog
- Complex Exception handling

AltFuture

- Thread group specified at creation
  - UI, WORKER, NET_READ, FLASH_WRITE
- Max concurrency contract
- Start order is FIFO..
  - ..then LIFO once a chain starts
- .onError(Exception) composition
  - _entire_ down-chain is notified

Tuned Thread Pools -> Heat Up The Phone

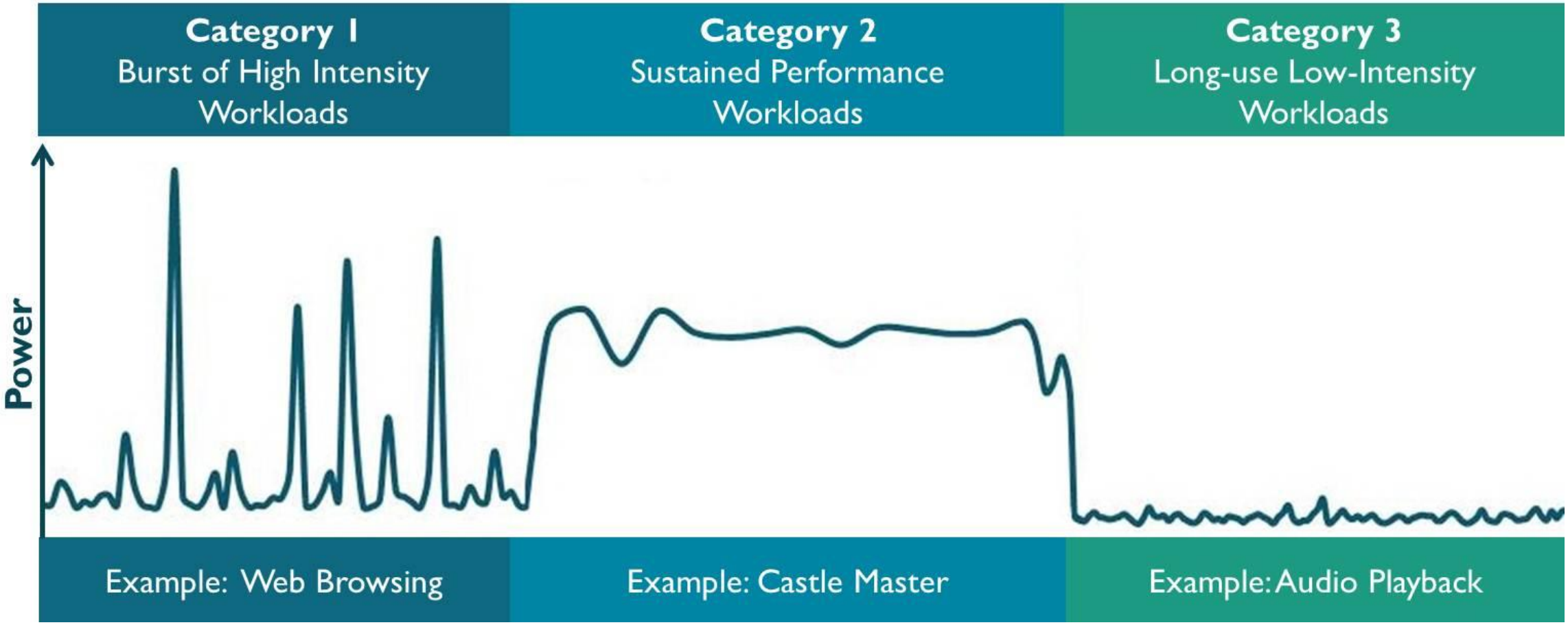


- Single Queue **per constraining resource**
  - Call each of these resource-defined queue+thread pool+tools an “aspect”
  - Logically easy
  - “Just right” concurrency for performance
  - Other categories like network, flash read/write also fight for CPU, but…
    - They use very little CPU, they use other chips in the phone
    - So these additional threads can be concurrent and “free speed” to the CPU

How do we make concurrent easy?

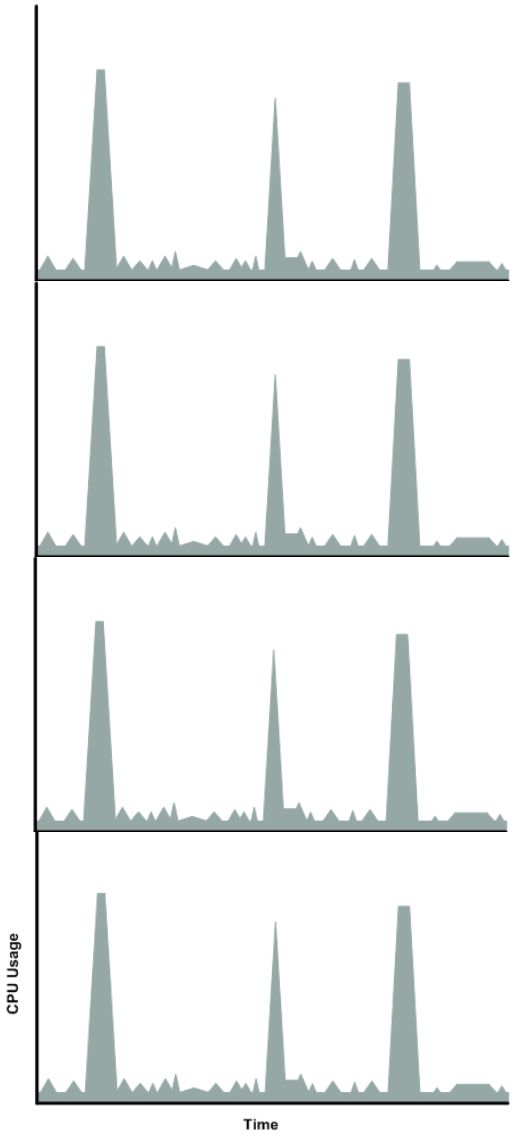
- Atomic + Functional
  - Very visible
    - heavily instrumented, runtime assertion checks, live graphic visualization, descriptive error messages, named objects and messages for readability, origin tracking and stack trace parsing
  - Atomic value objects are written for you
  - Concurrent reactive collections are written for you
  - Dependencies are explicit in the chain
  - Each object knows where to run and when to be concurrent or not
    - You can hint if you want to override the default “continue on same Aspect”
  - Resources use is constrained by FIFO aspect throttles
    - And LIFO chain completion before starting new chains
    - Only if LIFO is allowed (not an in-order-execution Aspect)
  - Automatic reactive clean up- no more plumbing leaks
    - How? Double linked list with WeakReference downchain, strong upchain
    - The reactive chain tail wags the dog. If gc() the tail, gc() upchain links until a current .split()
    - Beware impure reactive functions in the chain such as closures to long lived objects

Which workload do mobile apps have?



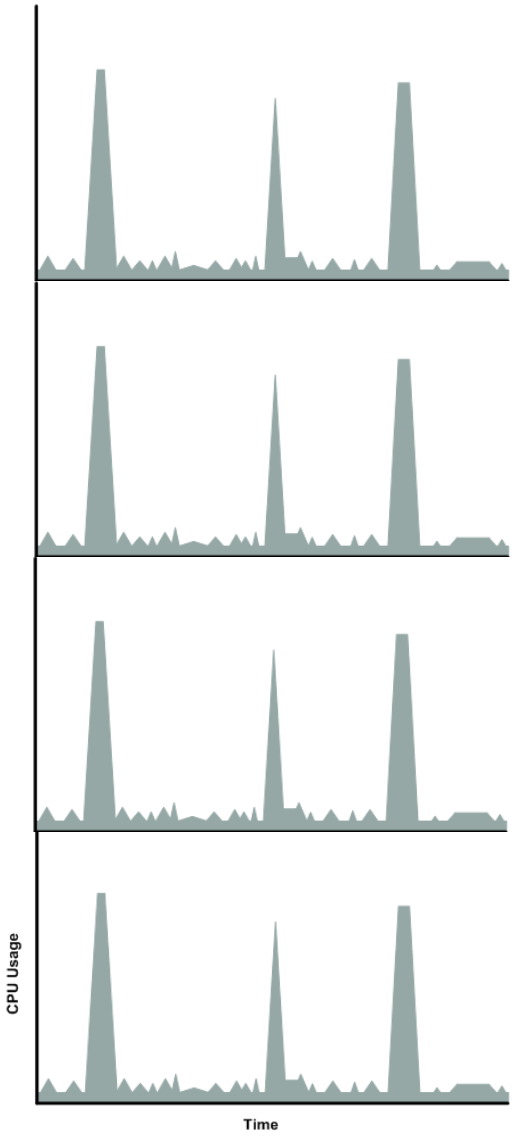
Network Contention Trade-offs and UX

- Finish connection A before starting B to keeps UI responsive
  - But.. TCP throttles up bandwidth use slowly per connection
- The roundtrip latency of starting connection B after A ends is dead air
  - Unless you start B while A is still ongoing
  - Precise, measure B onset time based on tuned filter prediction of past connection latency to a given server and tuned filter predicted A end time is possible. It has be done… but perhaps not worth it except on 2G
  - Simpler: have 1 more connection than you need, let chaos solve it
- Total memory load
  - Too many concurrently downloading byte arrays causes peak resource load



Why do you care that it is mostly spikes?

- 4-8 cores all working makes the spike narrower
- Narrow spikes -> UX
- Narrow spikes -> power savings
- Sequence first things first
  - Tasks not visible to users can be between spikes
  - So: less contention with UX during the spike



How many threads should you light up?

- UI
  - One thread to rule them all, one thread to..
- WORKER
  - Same as core count
- FLASH
  - READ: one (writes are cached, jump past queue)
  - WRITE: one (per flash device, depending on hardware)
- NET
  - READ
    - 2G: 2
    - 3G: 3
    - 4G & WIFI: 4
  - WRITE
    - Just 1
    - Asymmetric. You want to finish A before starting B

Iterate, RX, Cascade

Iterable pull

T next()

throws E

(+ plumbing)

RX

.onNext(T)

.onError(T)

.onCompleted()

(+ plumbing)

(+ baked methods)

(+ cleanup)

Cascade functional

.then(t -> f(T))

or

.then(t -> R f(T))

.onError(e -> f(E)

or

.onError((e, T) -> f(E, T)

AltFuture.fork()

SettableAltFuture.set(T)

Cascade reactive

.subscribe(t -> f(T))

or

.subscribe(t -> R f(T))

.onError(e -> f(E)

or

.onError((e, T) -> f(E, T)

AtomicValue.set(T)

ReactiveTextView.subscribe(String)

Iterate, RX, Cascade

- Finite sequence
- Infinite stream

Iterable pull

T next()

throws E

(+ plumbing)

- Finite sequence
- Infinite stream

RX

.onNext(T)

.onError(T)

.onCompleted()

(+ plumbing)

(+ baked methods)

(+ cleanup)

- Finite sequence

Cascade functional

.then(t -> f(T))

or

.then(t -> R f(T))

.onError(e -> f(E)

or

.onError((e, T) -> f(E, T)

AltFuture.fork()

SettableAltFuture.set(T)

- Infinite stream

Cascade reactive

.subscribe(t -> f(T))

or

.subscribe(t -> R f(T))

.onError(e -> f(E)

or

.onError((e, T) -> f(E, T)

AtomicValue.set(T)

ReactiveTextView.subscribe(String)

LambdaTest - [C:\Users\phou\android\cascade\LamdaTest] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 1.1.0

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

LamdaTest > app > src > main > res > layout > activity_main.xml

Android

- app
  - manifests
  - java
    - com.futurice.phou.lambdatest
      - MainActivity
    - com.futurice.phou.lambdatest (androidTest)
  - res
    - drawable
    - layout
      - activity_main.xml
    - menu
    - mipmap
    - values
- cascade
  - manifests
  - java
    - com.futurice.cascade
      - functional
      - i
      - reactive
        - ui
          - AltArrayAdapter
          - MirrorArrayAdapter
          - ReactiveEditText
          - ReactiveImageView
          - ReactiveTextView
        - AbstractSubscription

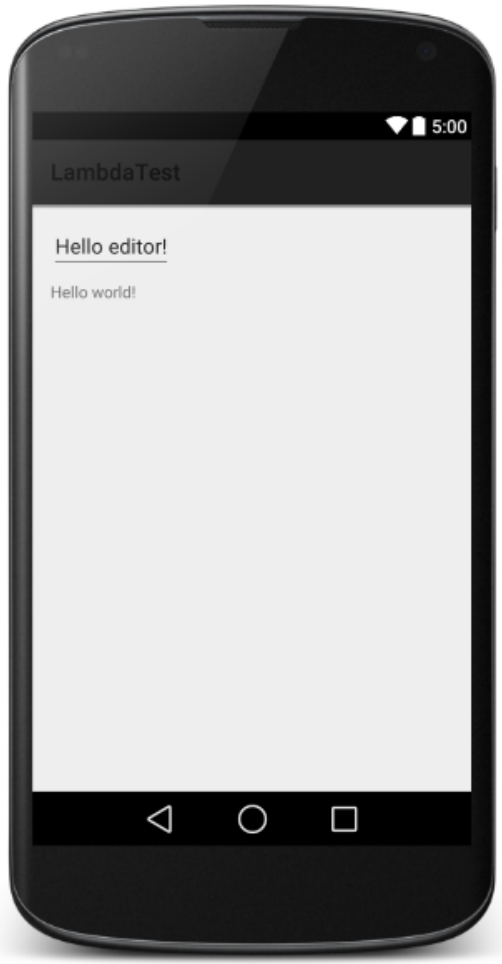
cascade × | MainActivity.java × | activity_main.xml ×

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <com.futurice.cascade.reactive.ui.ReactiveEditText
        android:text="Hello editor!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText" />

    <com.futurice.cascade.reactive.ui.ReactiveTextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText"
        android:id="@+id/textView"
        android:layout_marginTop="10dp" />


</RelativeLayout>
```

Design | Text

Preview

Nexus 4 | AppTheme | MainActivity | 22



TODO | 6: Android | Terminal | 0: Messages | Event Log | Gradle Console | Memory Mo

Gradle build finished in 11 sec (56 minutes ago)

16:1 CRLF UTF-8

```java
public class MainActivity extends ActionBarActivity {
    private static Async async;
    private static ReactiveValue<String> typedText;
    private static ReactiveValue<String> chainedText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (async == null) {
            async = new AsyncBuilder(this).build();
            typedText = new ReactiveValue<>(WORKER, "some_text_ReactiveValue");
            chainedText = new ReactiveValue<>(WORKER, "text_editor_ReactiveValue");
        }

        setContentView(R.layout.activity_main);

        ((ReactiveEditText) findViewById(R.id.editText)).subscribe(typedText);
        typedText.subscribe((String s) -> {
            return s.toLowerCase();
        })
            .subscribe(chainedText);
        chainedText.subscribe((ReactiveTextView) findViewById(R.id.textView));
```

Test - [C:\Users\phou\android\cascade\LamdaTest] - [app] - ...\app\src\main\java\com\futurice\phou\lambdatest\MainActivity.java - Android Studio 1.1.0

vigate Code Analyze Refactor Build Run Tools VCS Window Help

app

pp src main java com futurice phou lambdatest MainActivity

MainActivity.java × AbstractSubscription.java × activity_main.xml ×

```
        chainedText = new ReactiveValue<>(WORKER, "text_editor_ReactiveValue");
    }
```

at ADB logs

Log level: Verbose

app: com.futurice.phou.lambdatest

Nexus_S_API_22 Android 5.1 (API 22)

- cess.**acore** (1463)
- cess.**media** (1696)
- .**calendar** (1924)
- .**defcontainer** (2191)
- .**deskclock** (1951)
- .**dialer** (2090)
- .**email** (1802)
- .**exchange** (1728)
- .inputmethod.**latin** (1543)
- .**keychain** (2072)
- .**launcher** (1867)
- .**managedprovisioning** (2110)
- .**mms** (2128)
- .**music** (2047)
- .**phone** (1674)
- .providers.**calendar** (1969)
- .**settings** (1689)
- .**systemui** (1446)
- .phou.**lambdatest** (2276)
- **o** (2031)

logcat

```
03-12 01:28:19.663    2276-2276/com.futurice.phou.lambdatest V/AsyncBuilder:  Creating default serial worker executor service
03-12 01:28:19.663    2276-2276/com.futurice.phou.lambdatest D/AsyncBuilder:  Creating default in-order worker queue
03-12 01:28:19.674    2276-2276/com.futurice.phou.lambdatest D/AsyncBuilder:  setSerialWorkerQueue([])
03-12 01:28:19.675    2276-2276/com.futurice.phou.lambdatest V/AsyncBuilder:  setSerialWorkerExecutorService(java.util.concurrent.ThreadPoolExecutor@1f3b46d
03-12 01:28:19.676    2276-2276/com.futurice.phou.lambdatest V/AsyncBuilder:  setSerialWorkerAspect(com.futurice.cascade.DefaultAspect@1861e451)
03-12 01:28:19.676    2276-2276/com.futurice.phou.lambdatest D/AsyncBuilder:  setUiExecutorService(com.futurice.cascade.UIExecutorService@1ce514b6)
03-12 01:28:19.676    2276-2276/com.futurice.phou.lambdatest V/AsyncBuilder:  setUIAspect(com.futurice.cascade.DefaultAspect@149d75b7)
03-12 01:28:19.751    2276-2276/com.futurice.phou.lambdatest V/ReactiveValue-text_editor_ReactiveValue:  <UIAspect,UIThread> .onCreate(MainActivity.java:52)
    .onCreate(MainActivity.java:62)
    Adding WeakReference to down-chain IReactiveTarget "ReactiveTextView-2131296320" to "text_editor_ReactiveValue
03-12 01:28:19.753    2276-2276/com.futurice.phou.lambdatest V/ReactiveValue:  <UIAspect,UIThread> .onCreate(MainActivity.java:51)
    .onCreate(MainActivity.java:64)
    name=some_text_ReactiveValue Successful set(), value=jo jo jo
03-12 01:28:19.756    2276-2276/com.futurice.phou.lambdatest V/ReactiveValue-some_text_ReactiveValue:  <UIAspect,UIThread> .onCreate(MainActivity.java:51)
    .onCreate(MainActivity.java:64)
    fire(jo jo jo)
03-12 01:28:19.760    2276-2292/com.futurice.phou.lambdatest V/ReactiveValue-some_text_ReactiveValue:  <WorkerAspect,WorkerThread0> .onCreate(MainActivity.ja
    .doAction(AbstractSubscription.java:241)
    doReceiveFire "some_text_ReactiveValue value=jo jo jo
03-12 01:28:19.761    2276-2292/com.futurice.phou.lambdatest V/ReactiveValue-some_text_ReactiveValue:  <WorkerAspect,WorkerThread0> .onCreate(MainActivity.ja
    .doDownchainActions(AbstractSubscription.java:258)
    Fire down-chain reactive targets, but there are zero targets for some_text_ReactiveValue, value=jo jo jo
03-12 01:28:19.776    2276-2293/com.futurice.phou.lambdatest D/OpenGLRenderer:  Use EGL_SWAP_BEHAVIOR_PRESERVED: true
03-12 01:28:19.778    2276-2276/com.futurice.phou.lambdatest D/:  HostConnection::get() New Host Connection established 0xb3ee9e20, tid 2276
03-12 01:28:19.782    2276-2276/com.futurice.phou.lambdatest D/Atlas:  Validating map...
03-12 01:28:19.861    2276-2293/com.futurice.phou.lambdatest D/:  HostConnection::get() New Host Connection established 0xb3ffa150, tid 2293
03-12 01:28:19.866    2276-2293/com.futurice.phou.lambdatest I/OpenGLRenderer:  Initialized EGL, version 1.4
03-12 01:28:19.890    2276-2293/com.futurice.phou.lambdatest D/OpenGLRenderer:  Enabling debug mode 0
03-12 01:28:19.899    2276-2293/com.futurice.phou.lambdatest W/EGL_emulation:  eglSurfaceAttrib not implemented
03-12 01:28:19.899    2276-2293/com.futurice.phou.lambdatest W/OpenGLRenderer:  Failed to set EGL_SWAP_BEHAVIOR on surface 0xb3ff76e0, error=EGL_SUCCESS
```

Commander Maven Projects Gradle

What Next?

- MirrorService
  - REST with reactive concurrent set logic
  - MVVM example:

```
restWebServer.subscribe((JSON item) -> {
        return parse(item);
}
.subscribe(model)
.subscribe((String item) -> {
    return item.contains(filter);    // Filter is a reactive AtomicValue<String>
}
.subscribe(filteredViewModel);
```

What Next?

- Internet of Things cross-device closed loop lambda throw
  - Throw lambda logic to a device nearby
  - Wearable example:

```
restWebServer.subscribe((JSON item) -> {
        return parse(item);
}
.subscribe(commandModel)
.subscribe(WATCH, (String item) -> {
        return myWatch.process(item);
}
.subscribe(watchCommandResultsModel);
```

What Next?

- Information-centric network local resolution and actuators
  - Throw/receive lambda logic to/from devices by name (not net location)
  - Sensor example:

```
icn(“airTemperature”).subscribe((float temp) -> {
        return threeMinuteRollingAverage(temp);
}
.subscribe((float average) -> {
        system.println(“Nearby it is “ + average);
}
.subscribe(average ->
        icn(“bigScreenDisplay”).post(average))
.onError(() ->
        icn(“bigScreenDisplay”).post(“Problem getting temperature”);
```

What Next?

- Web app integration with local and remote POST services
  - GET, PUT, POST, DELETE by URL
  - Image repository example:

```
PostService.resolve(“http://stocks.com/api?ticker=msft”).subscribe((JSON priceJson) -> {
        …
}
```

Github

- https://github.com/paulirotta/cascade
- “No future but what we make” –John Connor, Terminator
- “Men have become the tools of their tools” –Henry David Thoreau
- “An apprentice carpenter may want only a hammer and saw, but a master craftsman employs many precision tools.

Or be common, follow the herd..


Developers take their tools into battle..

..with predicable results

FIGURE 1