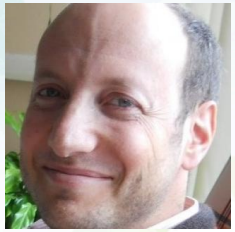
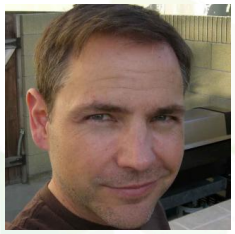
4 Mini Talk of 10 minutes each



Deterministic testing in a non deterministic world
Arturo Falck



Typesafe Config on Steroids
Eric Pederson

Hash Spreads and Probe Functions: A Choice of Performance and Consistency
Mohammad Rezaei

Real-Time Distributed Event-Driven Computing at Credit Suisse
Bill Brodie

QCon

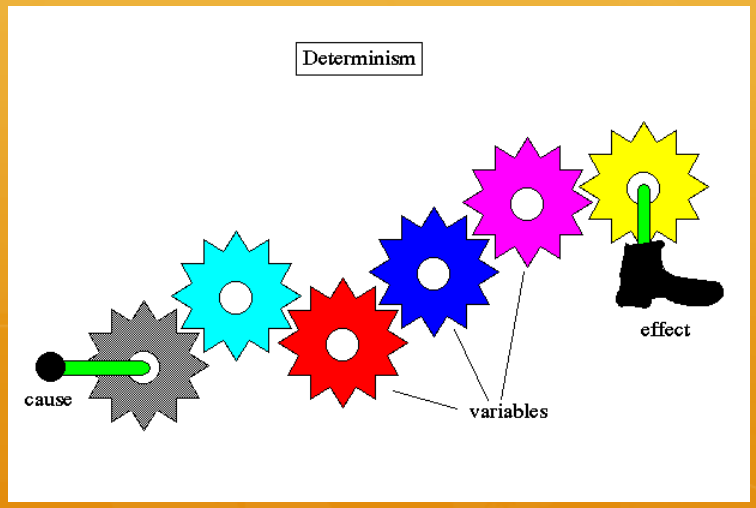
Deterministic testing
in a less deterministic world

Definitions

- Determinism:
- Random:
- Race Condition:

Definitions

- **Determinism:**
  The philosophy that everything has a cause, and that a particular cause leads to a unique effect.



- **Random:**
- **Race Condition:**

Definitions

- **Determinism:** .
- **Random:** lack of pattern or predictability in events
- **Race Condition:**

Definitions

- **Determinism:**
- Pseudo**random:** Algorithm that generates approximately random #s.
- **Race Condition:**

Definitions

- **Determinism:**
- Pseudo**random:**
- **Race Condition:** The output is dependent on the sequence or timing of other uncontrollable events.

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced
  LocalTime

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced
  LocalTime and **Clock**

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

```
public void bestMethod(Clock clock){
    LocalTime now = LocalTime.now(clock);
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

Time then and now

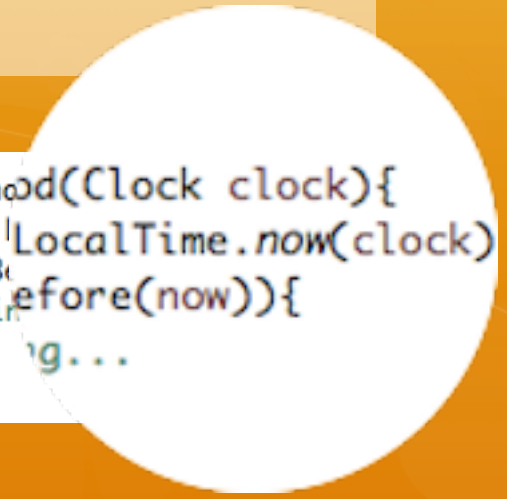
- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced
  LocalTime and **Clock**

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

```
public void bestMetho
    LocalTime now =
    if (DEADLINE.isB
        //do somethin
    }
}
```



Embrace Pseudorandom

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

Embrace Pseudo**random**

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

- **Better way:**
  Pass a **seed** to Random

```
public void betterRandom(long seed){
    Random die = new Random(seed);
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

Embrace Pseudo**random**

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

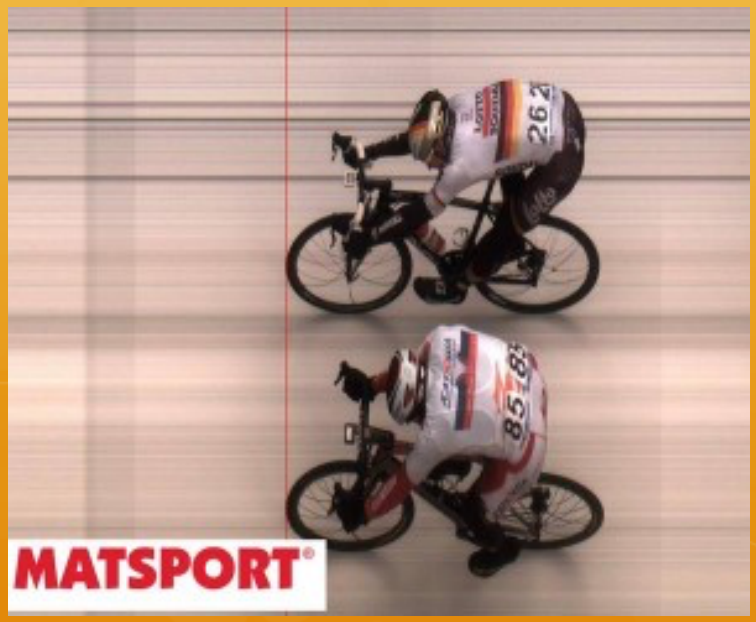
- **Better way:**
  Pass a **seed** to R

```
public void betterRandom(
    Random die = new Ra
    int roll = die.next
    switch(roll){
        case 0:  //do so
        case 1:  //do som
        case 2:  //do somet
        case 3:  //do somethin
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

```
random(long seed){
w Random(seed);
nextInt(6);
omething
```

Control **Race Conditions**

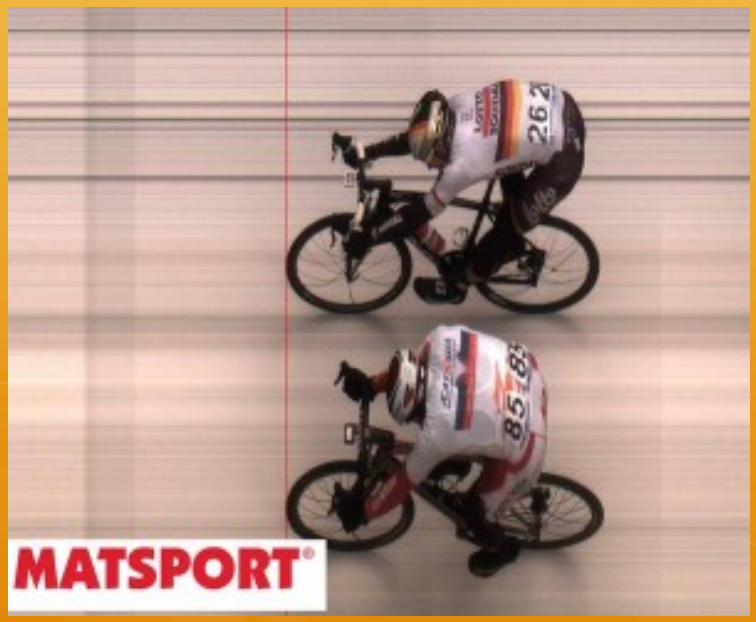
- **Thread Safety:**
  Many strategies to avoid corrupting data that is manipulated by multiple threads.
  Use:
  java.util.concurrent



- **Reproducible:**
- **Producer-Consumer:**

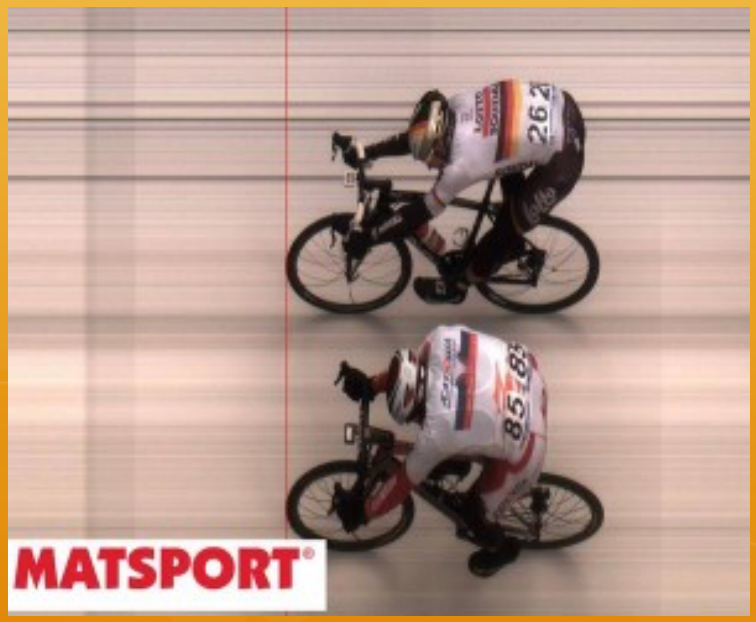
Control **Race Conditions**

- **Thread Safety:**
- **Reproducible:**
  More important for this discussion.
- **Producer-Consumer:**



Control **Race Conditions**

- **Thread Safety:**
- **Reproducible:**
- **Producer-Consumer:** One thread adds to a Queue, the other removes.



Control **Race Conditions**

- **Producer-Consumer:** Simple Example.

```
/** WARNING: do not use as synchronization monitor */
public class CommandManager {

    private List<Command> commandsForNextCycle = new LinkedList<Command>();

    public synchronized void add(Command command) {
        commandsForNextCycle.add(command);
    }

    public synchronized List<Command> processAll() {
        List<Command> commandsToReturn = commandsForNextCycle;
        //cutoff:  from this point on, we are collecting commands for the next cycle.
        commandsForNextCycle = new LinkedList<Command>();
        return commandsToReturn;
    }
}
```

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.

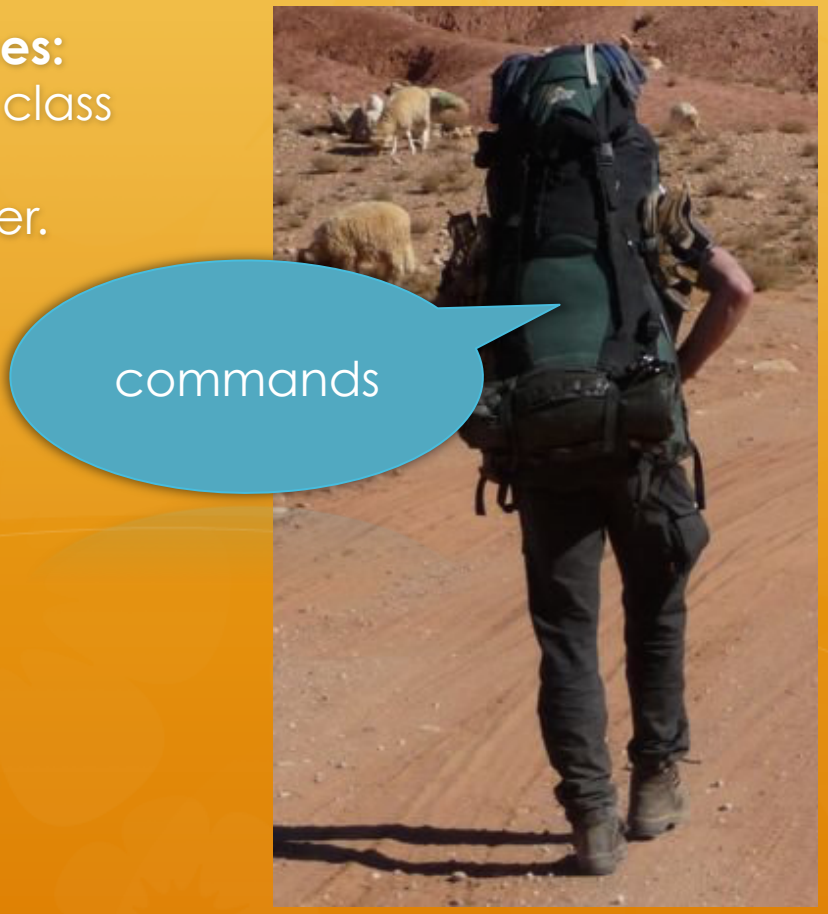
Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.

- **What Data?**
  Commands.

- **What Services?**
  Clock,
  Random Seed.

- **And…**
  Pass it around!

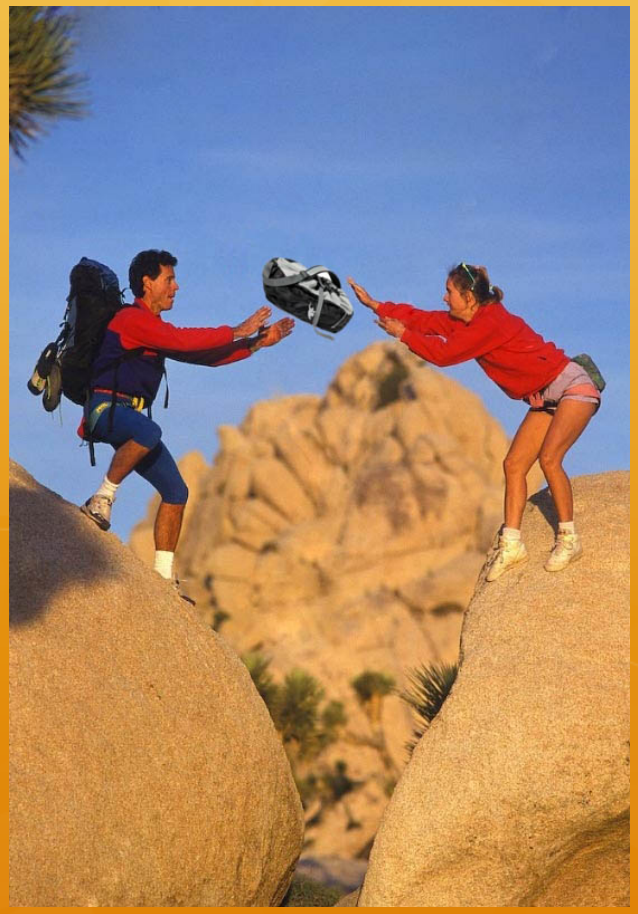


Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.
- **And…**
  Pass it around!

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.
- **And…**
  Pass it around!



Fleeting**Data**And**Services**

```
private FleetingDataAndServices(Clock clock, long randomSeed, List<Command> commands, Audi
    this.randomSeed = randomSeed;
    random = new Random(randomSeed);
    this.clock = Clock.fixed(clock.instant(), clock.getZone());
    this.commands = commands;
    this.auditor = auditor;
    this.tempo = tempo;
}
```

Pass it around!

Fleeting**Data**And**Services**

Pass it around!

Bring it all **Together**

- **log it** wisely

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice,

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice, testable

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice, testable system.

Deterministic testing
in a less deterministic world

Blog: **arturofalck.wordpress.com**
Code: github.com/afalck/offbeet-utils

Goldman Sachs

Hash Spreads and Probe Functions: A Choice of Performance and Consistency

GS.com/Engineering
March, 2015
Mohammad Rezaei, PhD

we BUILD

Goldman Sachs

Why is this important?

- Primitive collections have a significant advantage in memory and speed over boxed implementations.
- Hashing literature is mostly about hashing strings, not primitives.
- There are significant differences between different implementations.
  - Understanding the differences will empower you to pick the right solution.
- Benchmarking hash implementations is very hard, because it depends critically on the input.

Goldman Sachs

What is a hash spread?

- Patterns in input data can lead to too many collisions. E.g., all inputs are even.
- Collisions are always expensive, but more so in an open addressed hash structure.
- Hash spread is a function that destroys simple patterns in input data, while retaining maximal information about the input.
- There is a tradeoff between the complexity of the hash spread and how well the spread destroys patterns.

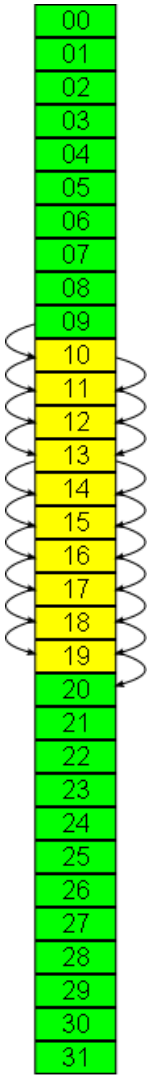
Goldman Sachs

What is a hash probe?

- In an open addressed hash structure, different array slots are examined when a collision occurs (no linked lists)
- The order of these slots must be deterministic and span the entire array. The order is called the probe function.
- Typical probe algorithms are linear, quadratic and "double hashing“
  - Double hashing uses $f(x) + n * g(x)$ and is the most expensive.
- Linear probing can lead to long chains and significant slow down.

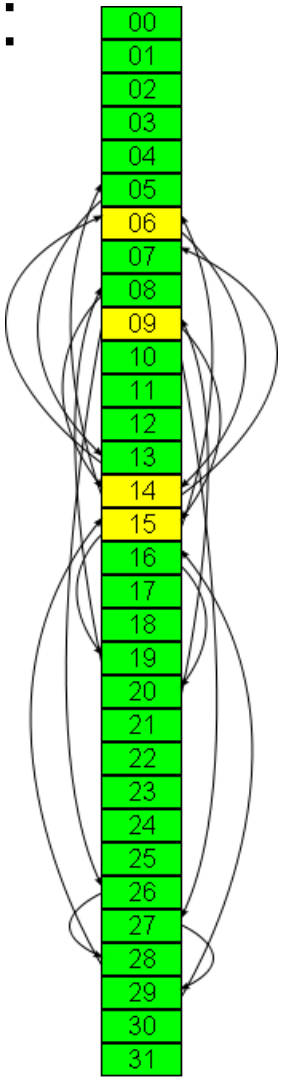
Linear and quadratic hash probes

Linear Probe:



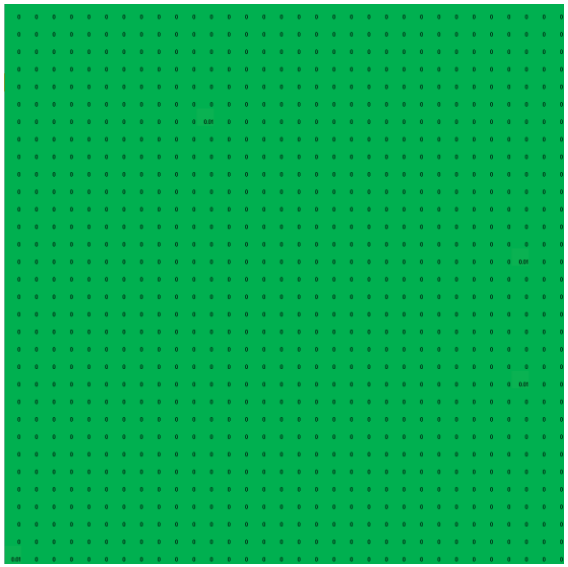
Quadratic Probe:

`17*n*(n+1)/2`

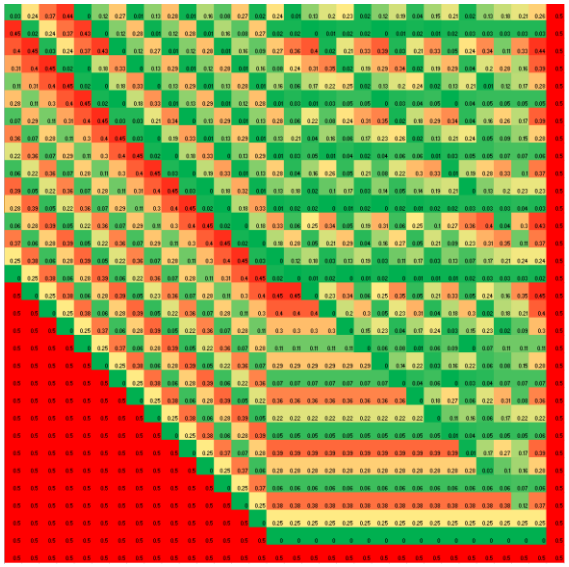


Comparing hash spreads

- Avalanche: probability of output bit changing when a single bit is flipped in the input. ideal is 50% for all input/output bits.
- GS Collections:



Koloboke:

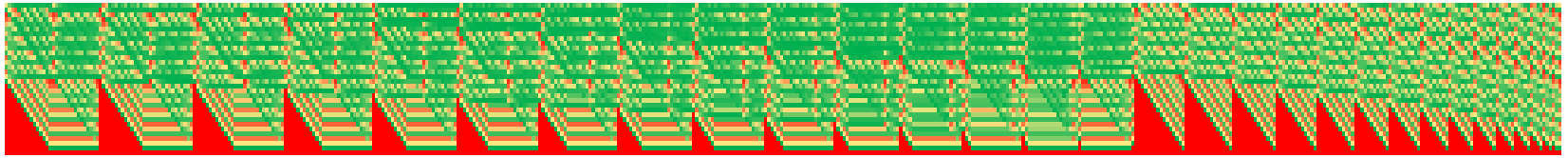


Goldman Sachs

Comparing hash spreads

- Bit independence criterion: output bits j and k should change independently when any single input bit i is inverted, for all i, j and k. (from wikipedia)
- GS Collections:

- Koloboke:



Goldman Sachs

Comparing implementations

- Koloboke uses a known deficient hash spread and linear probing.
  - It's easy to create slow structures.
- GS Collections 6.1 uses a good hash spread with quadratic probing.
  - Less likely than Koloboke to slow down, but quite a bit slower for the best case.

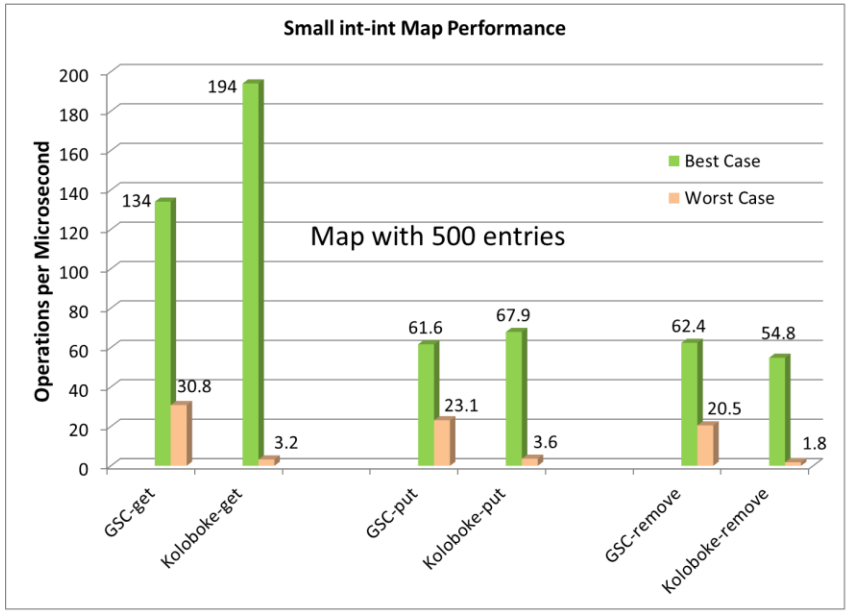
Goldman Sachs

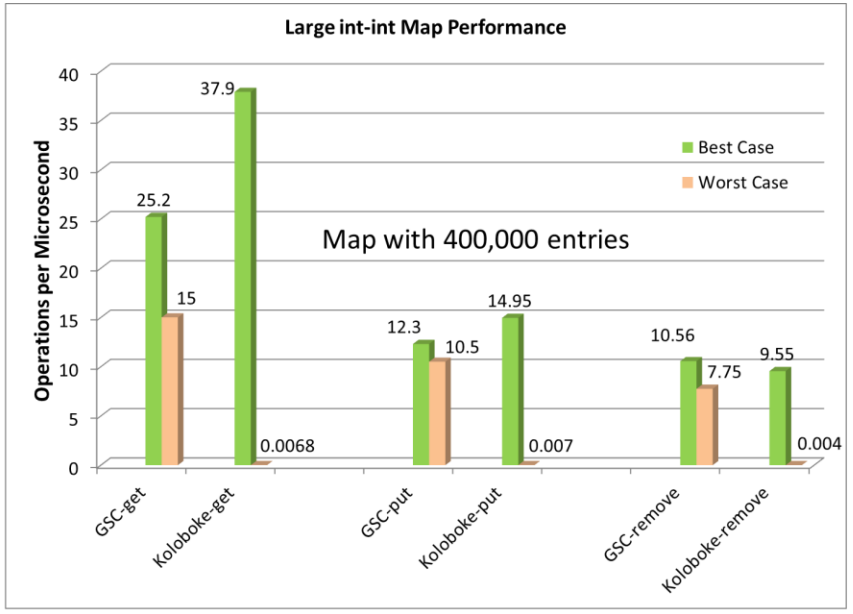
Comparing implementations

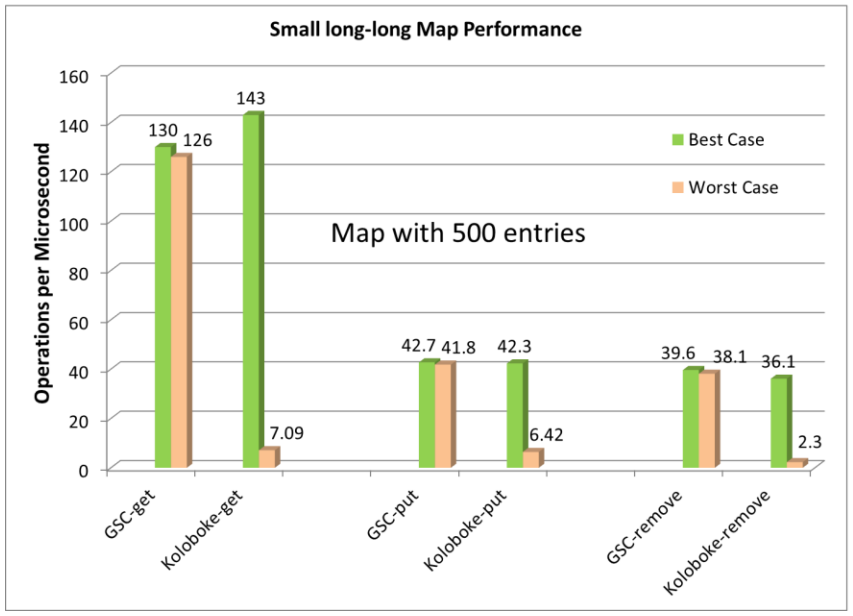
- GS Collections 6.2 introduces a hybrid algorithm that works well in best and worst case scenarios:
  - Start with no spread and a *short* linear probe (good for cache locality)
  - If first part of probe fails, then apply a good hash spread and do another short linear probe
  - If the second probe fails, use double hashing.

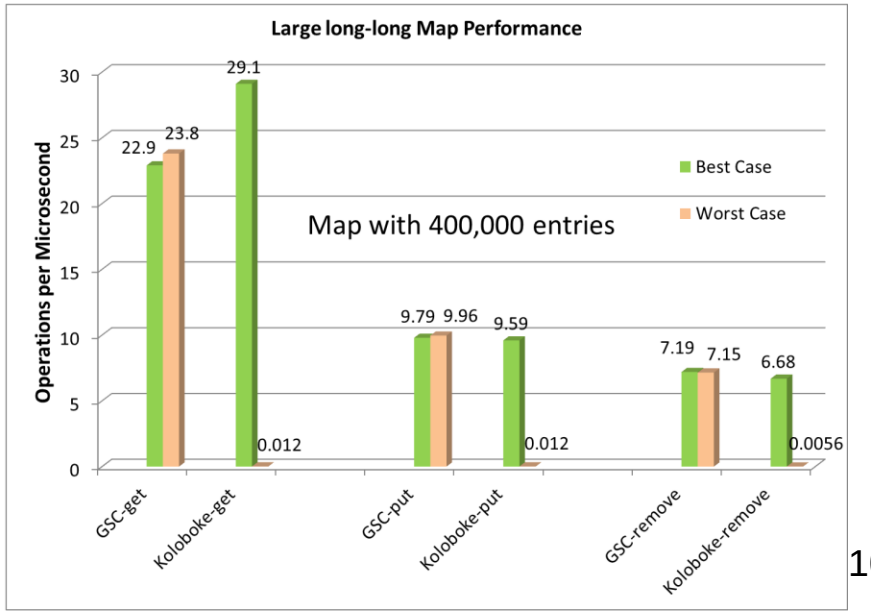
Goldman Sachs

Performance Comparison





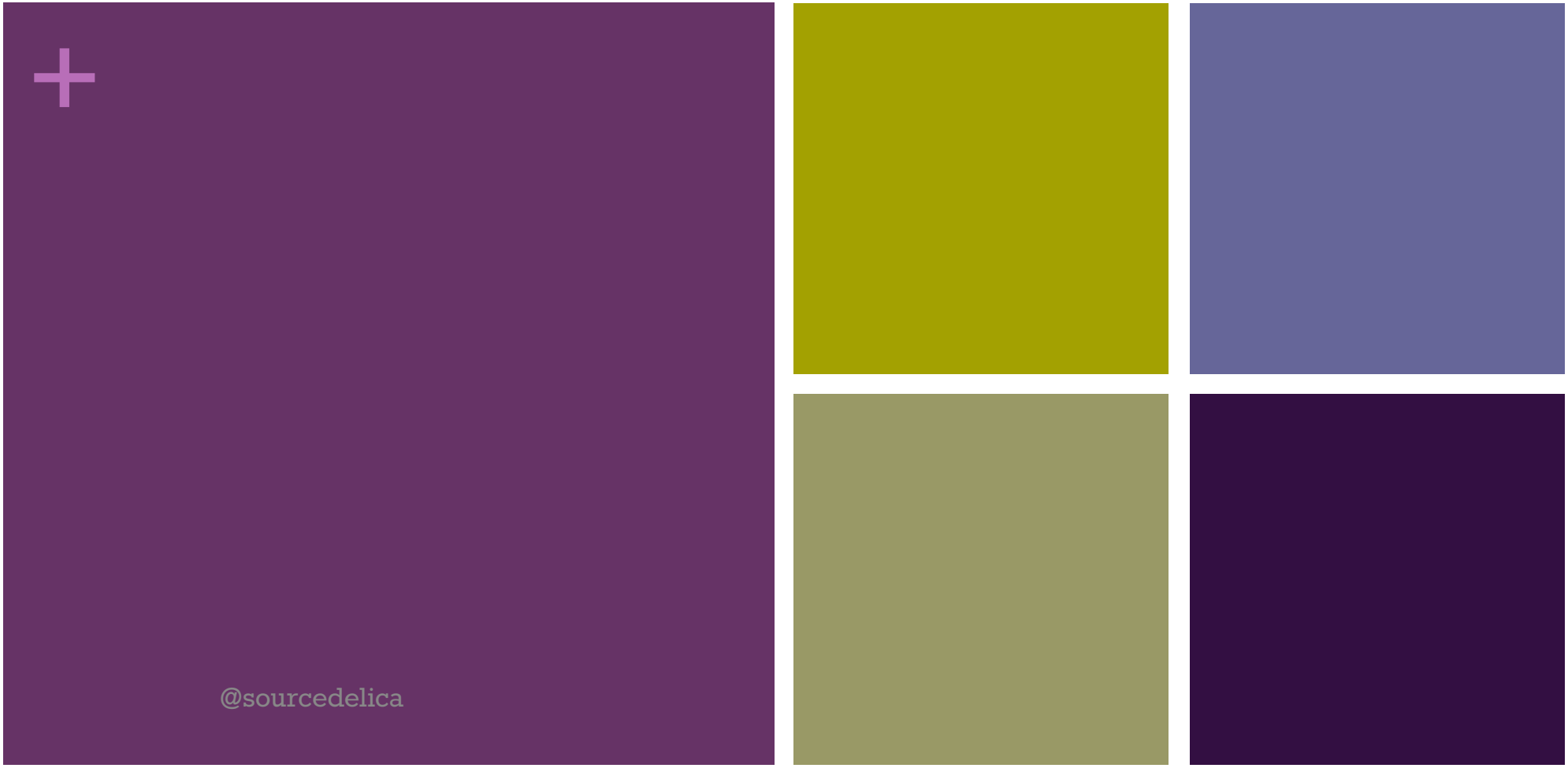




we

BUILD

Learn more at GS.com/Engineering
See the code at github.com/goldmansachs



Typesafe Config on Steroids

Agenda

- Introduction to Typesafe Config
- Scopes _ taking it to the next level

Property Files

- The old standby
  - On JVM
- Simple
- Hard to scale

```
#User properties modified on
#Tue Aug 24 17:49:35 CDT 2004
module.fileformat.FITS=ncsa.hdf.object.fits.FitsFile
module.fileformat.HDF=ncsa.hdf.object.h4.H4File
module.fileformat.HDF5=ncsa.hdf.object.h5.H5File
module.fileformat.Hdf-Eos2=hdfeos.he2.HE2File
module.fileformat.Hdf-Eos5=hdfeos.he5.HE5File
module.fileformat.NC=ncsa.hdf.object.nc2.NC2File
module.imageview=ncsa.hdf.view.DefaultImageView
module.metadataview=ncsa.hdf.view.DefaultMetaDataView
module.paletteview=ncsa.hdf.view.DefaultPaletteView
module.tableview=ncsa.hdf.view.DefaultTableView
module.textview=ncsa.hdf.view.DefaultTextView
module.treeview=ncsa.hdf.view.DefaultTreeView
recent.file0=E\:\\hdf-files\\SDSchunked.hdf
recent.file1=E\:\\hdf-files\\SDS_16_sziped.hdf
recent.file2=E\:\\hdf-files\\annras.hdf
recent.file3=E\:\\hdf-files\\h5-1km-szip-0.h5
recent.file4=E\:\\hdf-files\\h5-1km-gzip-3.h5
work.dir=E\:\\hdf-files
data.delimiter=Tab
extension.h4=hdf, h4, hdf4
extension.h5=hdf, h5, hdf5
file.extension=hdf, h4, hdf4, h5, hdf5, he5, he5
font.size=12
font.type=Dialog
max.members=10000
```

Apache Commons Configuration

- Typed API
- Substitutions
- Include other files
- Downsides
  - Limited by property file format
  - Composition is limited

Typesafe Config

- Used by Play Framework and Akka
  - But is a standalone project with no dependencies
- JSON-like format
- Java API
  - Typed
  - Immutable
  - Power features

Typesafe Config Format

- HOCON
  - Human-Optimized Config Object Notation

```
demo {
  # Our development DB
  database {
    username = "scott"
    password = "tiger"
    connections = 20
    timeout = 5 seconds
  }
  jms.brokers =
    ["nyamq10", "sfamq20", "jpamq54" ]
```

External Values

- System properties are used by default
  - Easy to override configuration values in scripts

- Reference environment variables

```
basedir = "/foo/bar/baz"
basedir = ${?BASEDIR_OVERRIDE}
```

  - Would override `basedir` if set

API examples

```
# Loads all application.conf in the classpath
Config config = ConfigFactory.load()

String username =
  config.getString("demo.database.username")

int connections =
  config.getInt("demo.database.connections")
```

API Examples

```
# Config values are like "15s", "5 minutes"
Duration timeout =
  config.getDuration("demo.database.timeout")

# Config values are like "100m", "2 gigs"
long size = config.getBytes("demo.cache.size")

List<String> brokers =
  config.getStringList("demo.jms.brokers")
```

Create Configs at Runtime

```
val config = ConfigFactory.parseString(
      s"""
        |akka {
        |  remote {
        |    netty.tcp {
        |      hostname = "$hostname"
        |      port = $port
        |    }
        |  }
        |}
      """.stripMargin)

val system = ActorSystem("demo", config)
```

Config Objects

```
demo {
  services = [
    { name = "user",
      host = "nycdev101", port = 8080
    }
    { name = "search",
      host = "nycdev102", port = 9201 }
    # etc...
  ]
}

List<Config> configList =
  config.getConfigList("demo.services")
```

Merging Configs

```
Config appConfig=
  ConfigFactory.parseFile("application.conf")

Config userConfig =
  ConfigFactory.parseFile(username + ".conf")

# Merge configs and resolve substitutions
Config finalConfig =
  ConfigFactory.systemProperties().
    withFallback(userConfig).
    withFallback(appConfig).
    resolve()
```

Scopes Library

- A library built on top of Typesafe Config
- Adds
  - Multiple scopes of configuration
    - aka Configurable config merging
  - Pluggable resource handling
  - and more

Scopes Configuration

```
scopes = [
    {
      name = application
      path = "classpath://config.conf"
    }
    {
      name = environment
      path = "classpath://config-${environment}.conf"
    }
    {
      name = shared
      path = "zk://config/shared"
    }
    {
      name = shared
      path = "zk://config/shared/${environment}"
    }
]
```

Scopes API

- Superset of Typesafe Config API
- Adds
  - Scala API
  - Resource values
  - Defaults
  - Dynamic updates
  - and more

Extended API

```
demo {
  jms.brokers =
    ["nyamq10", "sfamq20", "jpamq54" ]
}
```

```
val config = Scopes.config()

# Scala collections are used by default
config.getStringList("demo.jms.brokers").
  foreach(println)
```

Extended API

```
demo {
  database.username = "scott"
  database.password = "tiger"
}

val username =
  config.getString("demo.username", "admin")

# Returns Some[String] or None if not set
val password =
  config.optString("demo.password").
    getOrElse("admin")
```

Resource Properties

```
demo {
  # Currently supported: classpath, file, zk
  helpText = "classpath://help/help.txt"
}
```

```
val helpResource =
  config.getResource("demo.helpText")

val helpText = helpResource.asString

val helpStream = helpResource.asStream
```

Dynamic updates

- Watches config locations defined in scopes
- If changes are made, callbacks get new values

Interested?

- Interested in Scopes?
  - I'm thinking of open-sourcing
  - Contact me @sourcedelica

More Info

- Github
  - https://github.com/typesafehub/config
- Javadoc
  - http://typesafehub.github.io/config/latest/api/
- Intellij
  - Scala Plugin has editing support
    - Syntax checking, folding

CREDIT SUISSE

Architecture Spotlight:
Vertical Markets | The Consumer Web

Bill Brodie, Allium Technology

Preliminary – Not for Publication

June, 2015

Static Model

- Batch updates
- Standardized reports
- Real-time analytics

Dynamic Model

- Huge transaction volumes
- Data science and complex analytics produced in real time
- User-driven, on-demand reporting
- Network effects

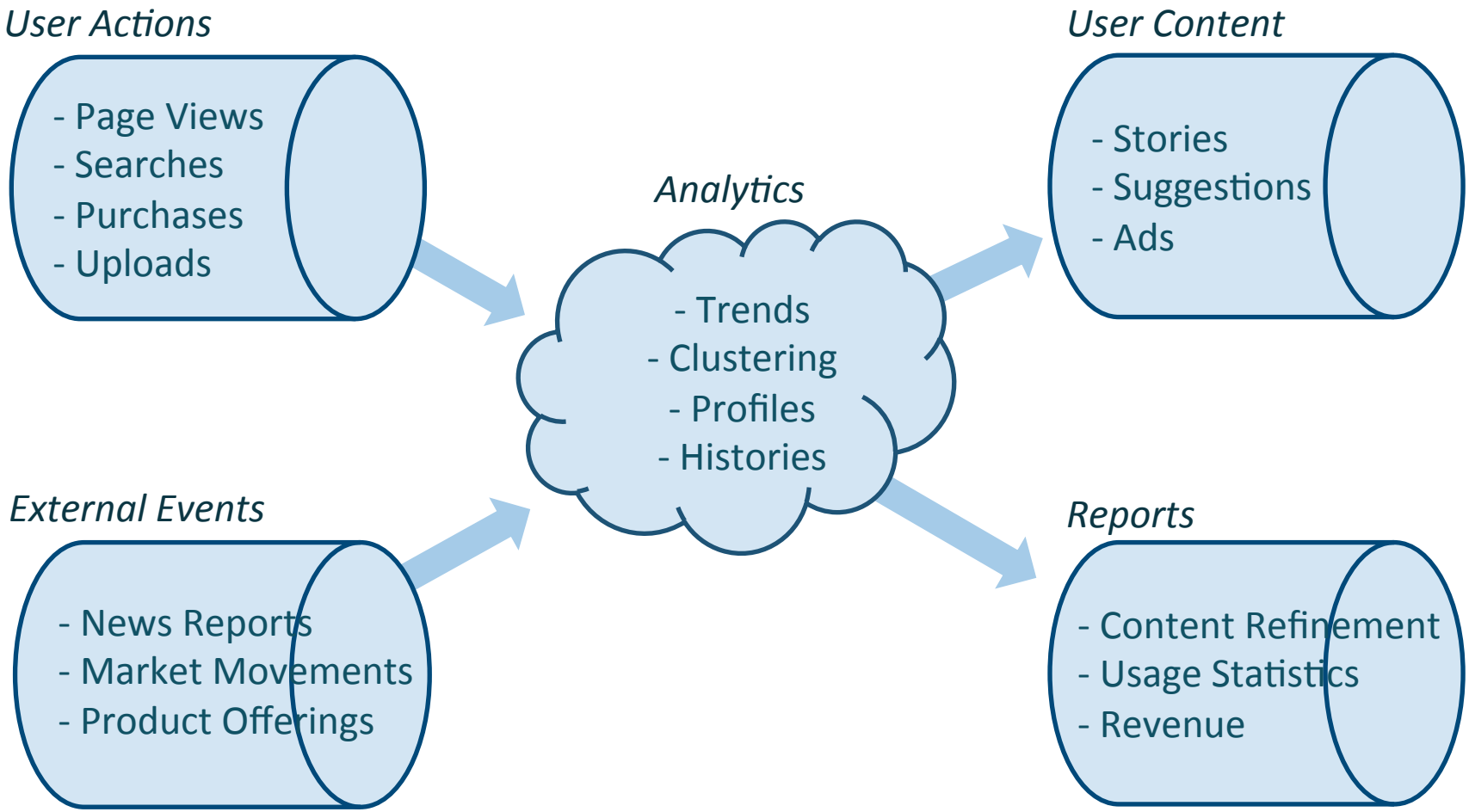
Pioneers

- Google
- Facebook
- Netflix
- Amazon

New Technologies

- NoSQL and graph databases
- Cloud computing
- Small, scalable servers
- Fault tolerance

Data Flow – Consumer Site



Data Flow – Finance

Live and Static Trade Data

ID	Book	Security	NbShares
	18579	ORCL	220
1	2142039	COST	8423
2	12145	ALTR	3520
3	21121	AKAM	10020
4	2036915	AAPL	3383
5	2055103	VMED	4954
6	15316	CMCSA	8461

Streaming Market Data

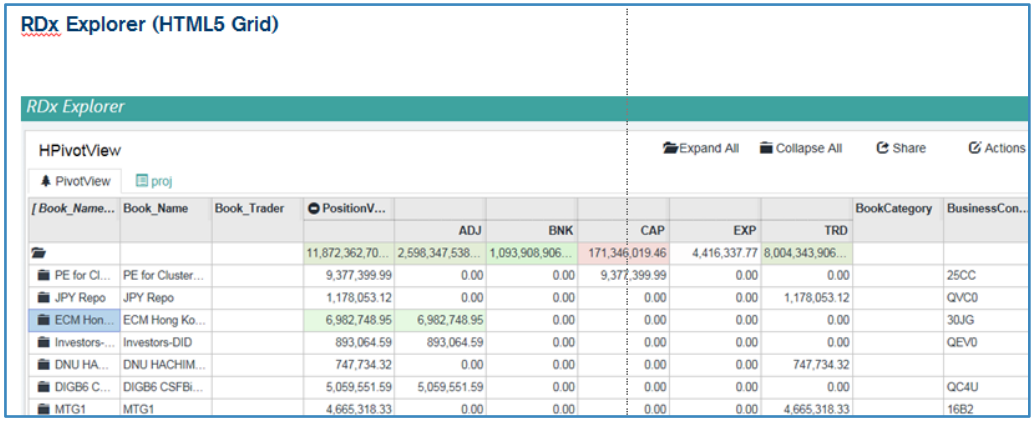
ID	Security	Bid	Ask
	ATVI	63.90	64.05
1	ADBE	14.55	16.52
2	AKAM	340.20	341.49
3	ALXN	5.37	6.86
4	ALTR	443.04	444.54
5	AMZN	59.82	60.94
6	AMGN	60.29	60.82

Static Reference Data

Big data sets, Live data, Complex logic

Huge user base, diverse needs

Live, rich, big interactive reports

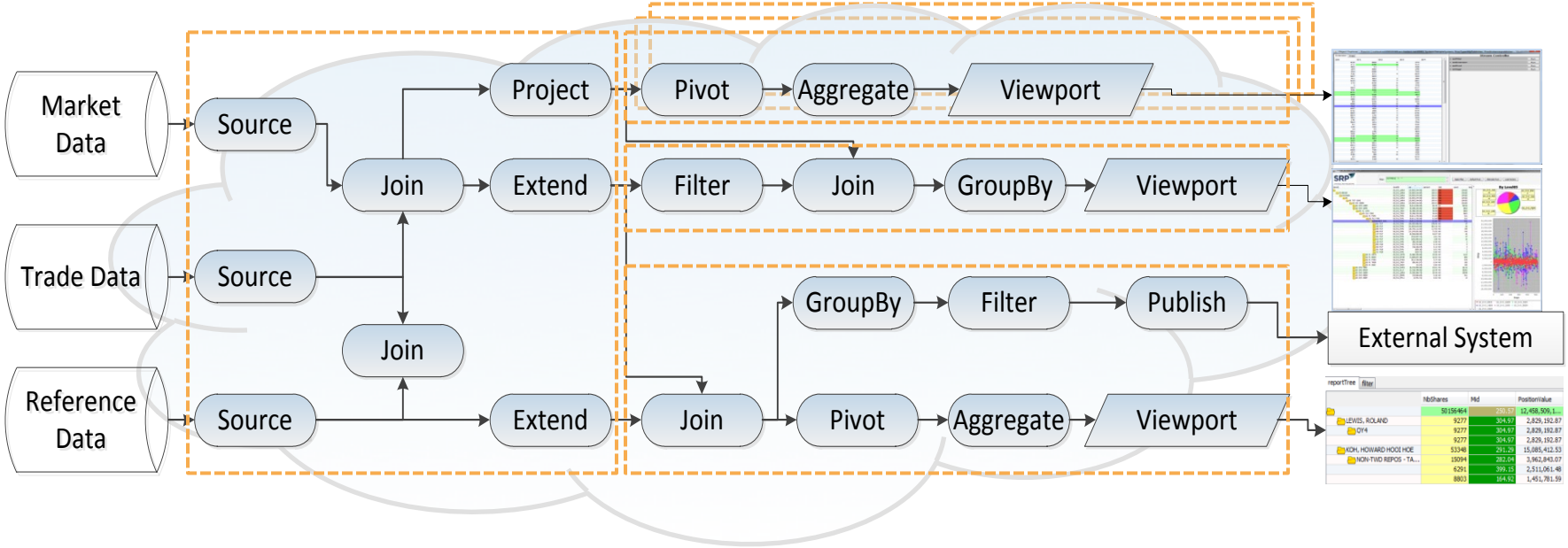


[Book_Name...	Book_Name	Book_Trader	PositionV...	ADJ	BNK	CAP	EXP	TRD	BookCategory	BusinessCon...
			11,872,362,70...	2,598,347,538...	1,093,908,906...	171,346,019.46	4,416,337.77	8,004,343,906...		
PE for Cl...	PE for Cluster...		9,377,399.99	0.00	0.00	9,377,399.99	0.00	0.00		25CC
JPY Repo	JPY Repo		1,178,053.12	0.00	0.00	0.00	0.00	1,178,053.12		QVC0
ECM Hon...	ECM Hong Ko...		6,982,748.95	6,982,748.95	0.00	0.00	0.00	0.00		30JG
Investors-...	Investors-DID		893,064.59	893,064.59	0.00	0.00	0.00	0.00		QEV0
DNU HA...	DNU HACHIM...		747,734.32	0.00	0.00	0.00	0.00	747,734.32		
DIGB6 C...	DIGB6 CSFBi...		5,059,551.59	5,059,551.59	0.00	0.00	0.00	0.00		QC4U
MTG1	MTG1		4,665,318.33	0.00	0.00	0.00	0.00	4,665,318.33		16B2

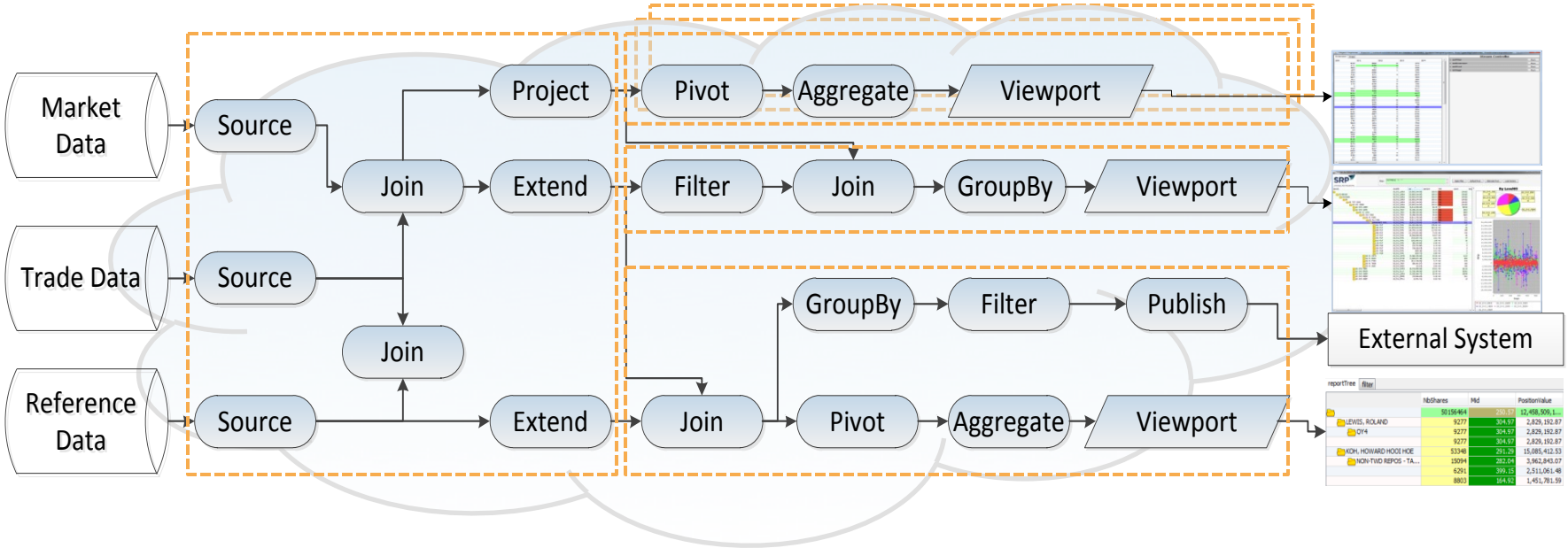
Platform Elements

- Data capture and audit
- Grid calculation
- Real-time data flow
- Visualization

Real-Time Data Algebra



Distributed Runtime



Records — **Typed tuples**
Stored as a Java class of primitive types (no boxing!) by synthesizing JVM byte code on-the-fly

Operators — **Highly optimized building blocks**
Build complex live systems by functional composition of algebraic operators

Expressions — **Java language support**
Predicates, expressions, aggregations can all be dynamically changed at runtime

Messaging — **Compact delta messaging**
Only changes are propagated thru the system using a very compact message format

Processes — **System-level deployment**
Global namespace, load-balancing, failover, and elasticity provided by design

Benefits of Common Platform

- Rapid development
- Handles infrastructure complexities
- Ensures consistency

Commonalities with massively scaled Web sites

- Volume & Velocity
- Real-time decision-making
- Dynamic, flexible views
- Graph-like data model
- Use of open-source tools
- Synthesis of real-time data from many sources
- Failover, load balancing, fault tolerance

Advantages over consumer sites

- Well-defined problem space
- Steady feedback leads to tailored refinement
- Demanding, insightful users
- Techniques translate directly to large set of applications across industries

Bill Brodie

wbrodie@alliumtechnology.com

+1 (212) 414-0403

Q&A