KEEP
CALM
AND
QUERY
ON

PROGRAMMING PRINCIPLES FOR A DISTRIBUTED ERA

JOE HELLERSTEIN
BERKELEY
TRIFACTA

OUTLINE BY ANALOGY

THE GLORY
OF
HISTORY

ELEGANTLY TENACIOUS

FRUSTRATINGLY PERSISTENT

http://www.theurbangent.com/2011/04/style-of-prince-william-of-wales-duke-of-cambridge.html

KEEP
CALM
AND
CARRY
ON

OUTLINE

HISTORY

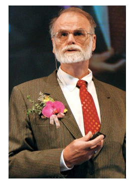
TENACITY

FRUSTRATION

CALM

OUTLINE

VON NEUMANN



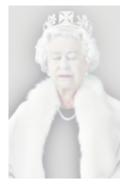
TRANSACTIONS

CAP

CALM

OUTLINE

HISTORY



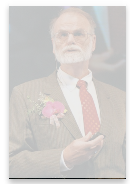
TENACITY

FRUSTRATION

CALM

OUTLINE

von NEUMANN



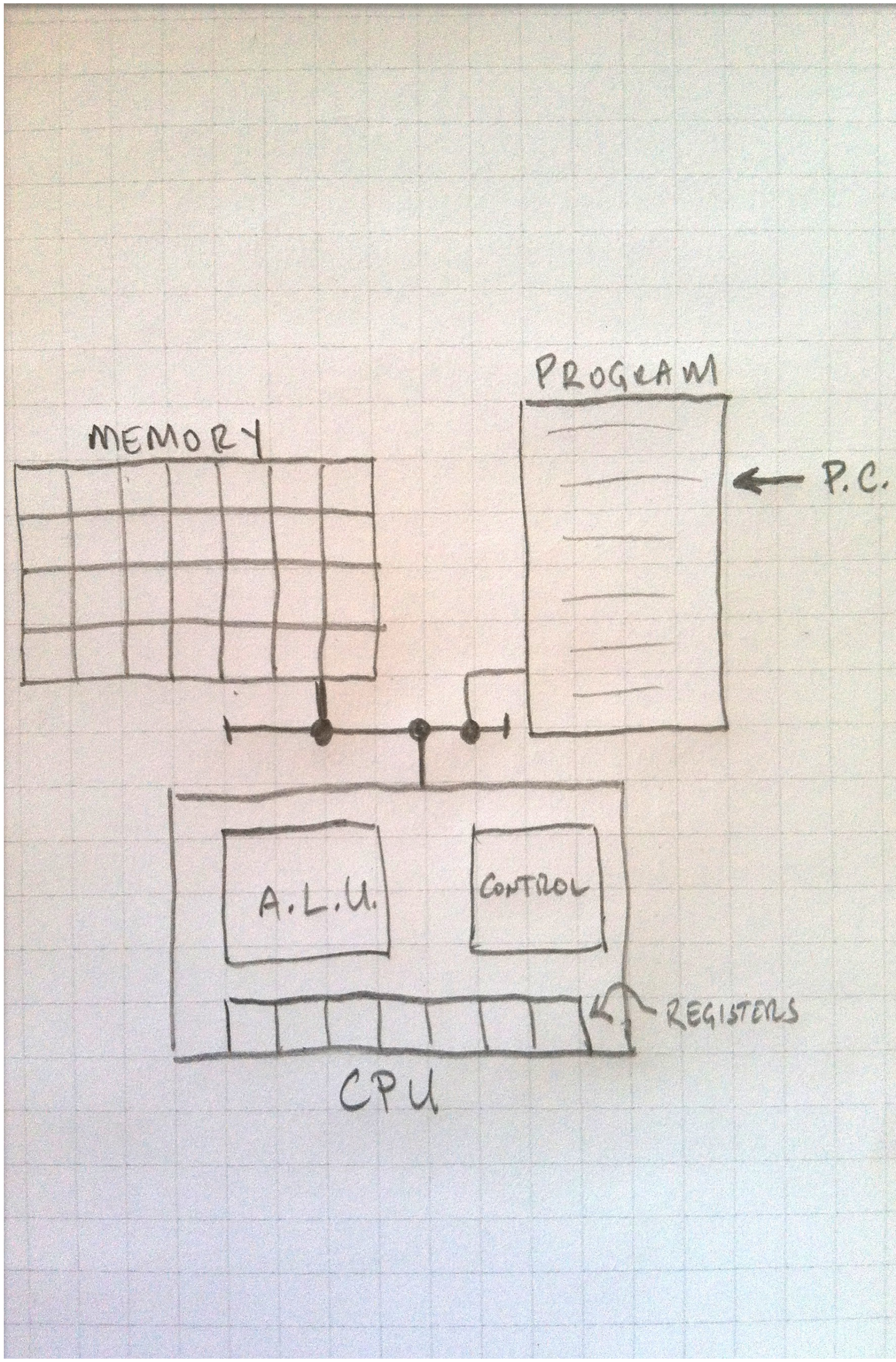
TRANSACTIONS

CAP

CALM

THE von NEUMANN MACHINE

- ORDER
  - LIST of Instructions
  - ARRAY of Memory
- THE STATE
  - Mutation in time

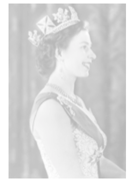


THE von NEUMANN MACHINE

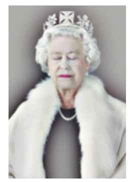
- ORDER
  - LIST of Instructions
  - ARRAY of Memory
- THE STATE
  - Mutation in time

ORDER
AND
THE STATE

OUTLINE



HISTORY

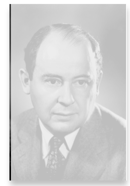


TENACITY

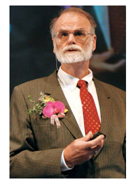
FRUSTRATION

CALM

OUTLINE



VON NEUMANN



TRANSACTIONS

CAP

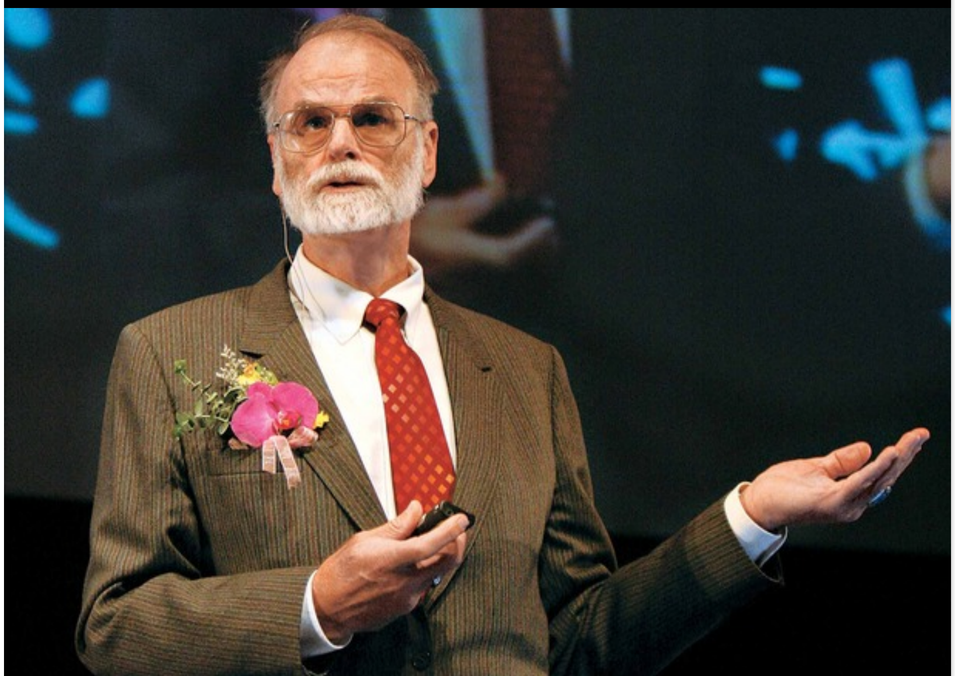
CALM

RACE

DISORDER

AND

THE STATE

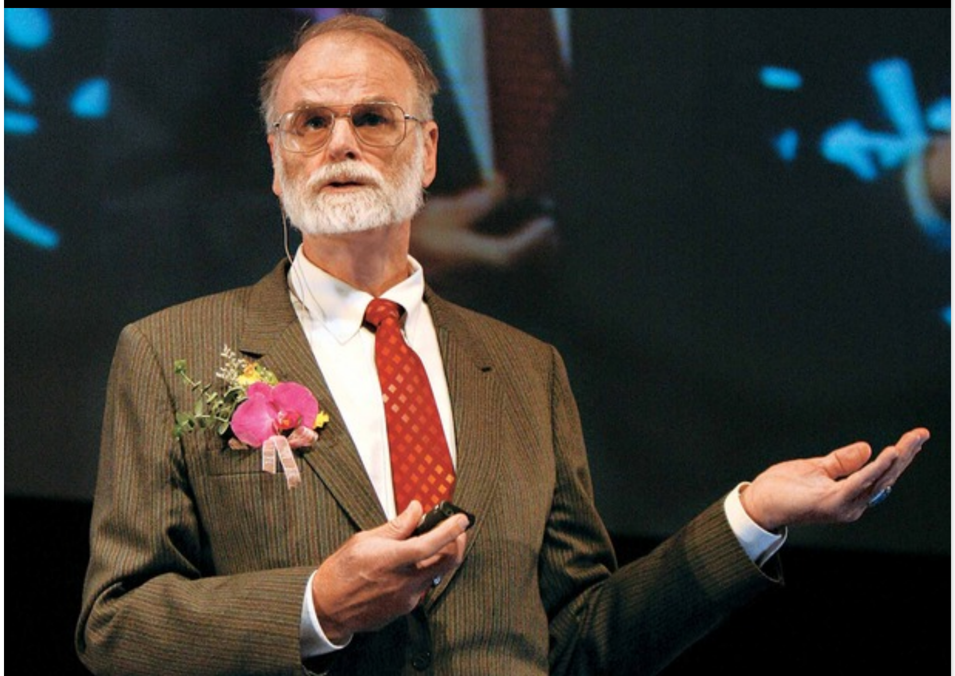


http://www.nytimes.com/2009/12/15/science/15books.html

THE TRANSACTION CONCEPT

- ORDER
  - Disorder *across* transactions
  - Illusion of order *within* transactions
- THE STATE
  - Registers, Memory
    - Isolation
  - Mutation in time
    - Atomicity

THE TRANSACTION CONCEPT

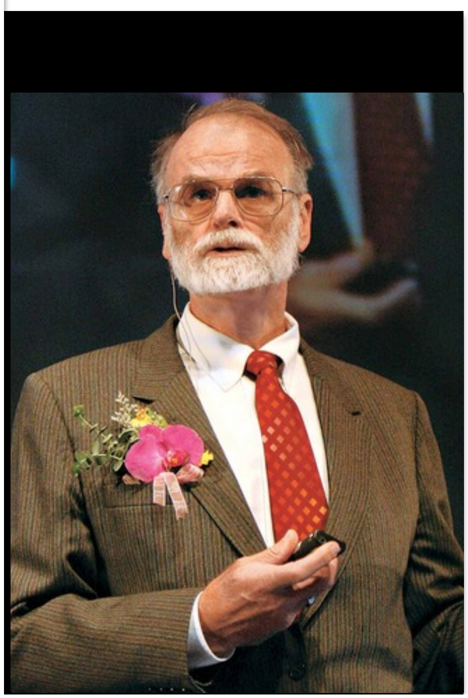


http://www.nytimes.com/2009/12/15/science/15books.html

- ELEGANT THEORY
  - Serializability
- PRACTICAL ENGINEERING
  - A transparent illusion
    - Easy to ensure correct applications
    - Tricky to scale infrastructure


SUMMARY

- TRIUMPH OF ORDER
- TRIUMPH OF THE STATE




- ELEGANT ILLUSION OF ORDER AND STATE
  - FORMAL THEORY
  - NATURAL API
  - EFFICIENT IMPLEMENTATION

OUTLINE

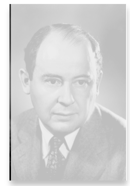
HISTORY

TENACITY

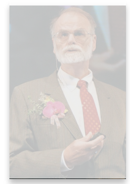
FRUSTRATION

CALM

OUTLINE



VON NEUMANN

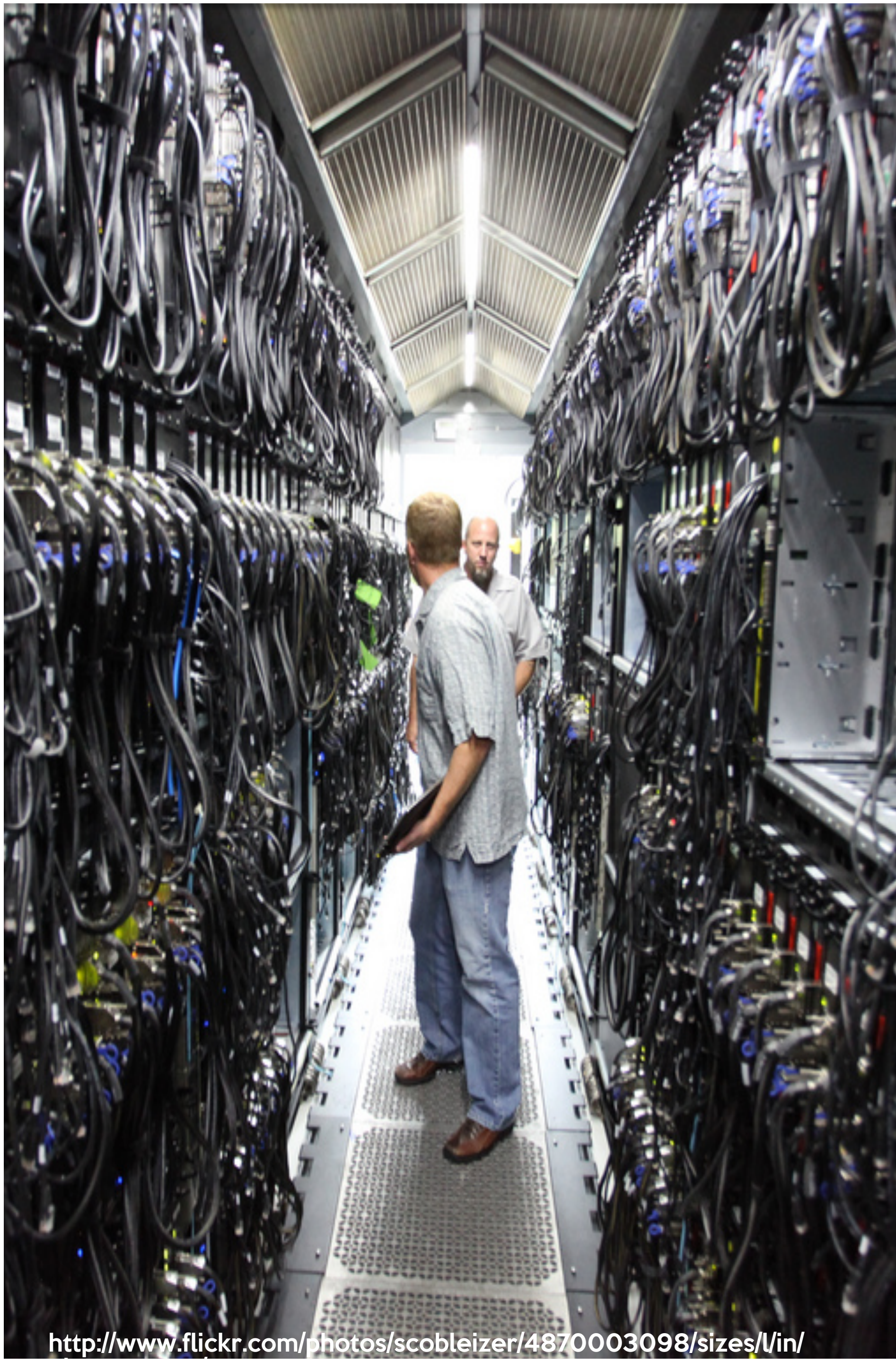


TRANSACTIONS

CAP

CALM

ORDER, THE STATE AND GLOBALIZATION



DISTRIBUTED COMPUTING IS THE NEW NORMAL

- ORDER IS TOO COSTLY
  - Synchronization
  - Coordination
- THE STATE IS HEARSAY
  - Delay
  - Failure
  - Partition

THE CAP THEOREM

THE
CAP
THEOREM

MIND THE CAP

A NEGATIVE RESULT

FOR A TIME OF

DISILLUSIONMENT

COPING
WITH
DISORDER

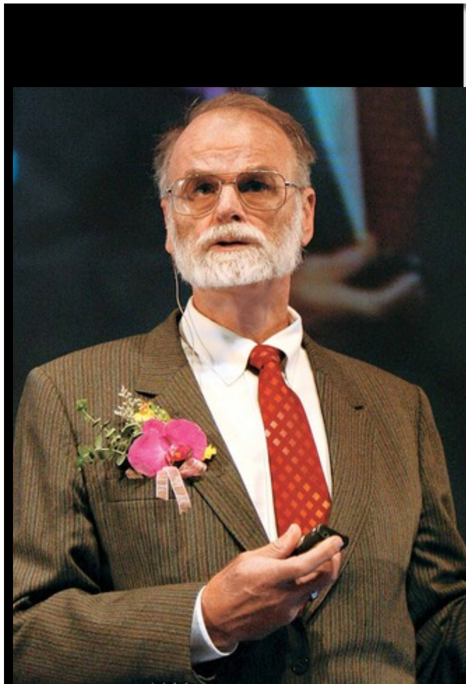
- **DESIGN MAXIMS**
  - Commutative methods
  - Inverse methods
  - Free coupons
- **PRACTICAL ENGINEERING**
  - Pragmatic Systems
    - Easy to scale infrastructure
    - Tricky to ensure correct applications

THE TRANSACTION CONCEPT

- ELEGANT THEORY
  - Serializability
- PRACTICAL ENGINEERING
  - A transparent illusion
    - Easy to ensure correct applications
    - Tricky to scale infrastructure

COPING WITH DISORDER

- DESIGN MAXIMS
  - Commutative methods
  - Inverse methods
  - Apologies
- PRACTICAL ENGINEERING
  - Pragmatic Systems
    - Easy to scale infrastructure
    - Tricky to ensure correct applications



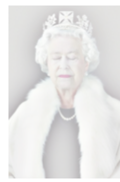
SUMMARY

ELEGANCE & ORDER
EXPENSIVE ILLUSIONS

MAXIMS & DISORDER
FRAGILE APPLICATIONS

OUTLINE

HISTORY

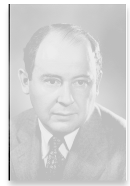


TENACITY

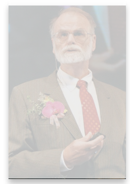
FRUSTRATION

CALM

OUTLINE



von NEUMANN



TRANSACTIONS

CAP

CALM

KEEP
CALM
AND
QUERY
ON

POSITIVE THINKING
FOR THE
CLOUDY FUTURE

THE TRANSACTION CONCEPT

- ELEGANT THEORY
  - Serializability
- PRACTICAL ENGINEERING
  - A transparent illusion
    - Easy to ensure correct applications
    - Tricky to scale infrastructure

COPING WITH DISORDER

- DESIGN MAXIMS
  - Commutative methods
  - Inverse methods
  - Apologies
- PRACTICAL ENGINEERING
  - Pragmatic Systems
    - **Easy** to scale **infrastructure**
    - **Tricky** to ensure correct **applications**

CALM

THEORY

FOR

APPLICATIONS

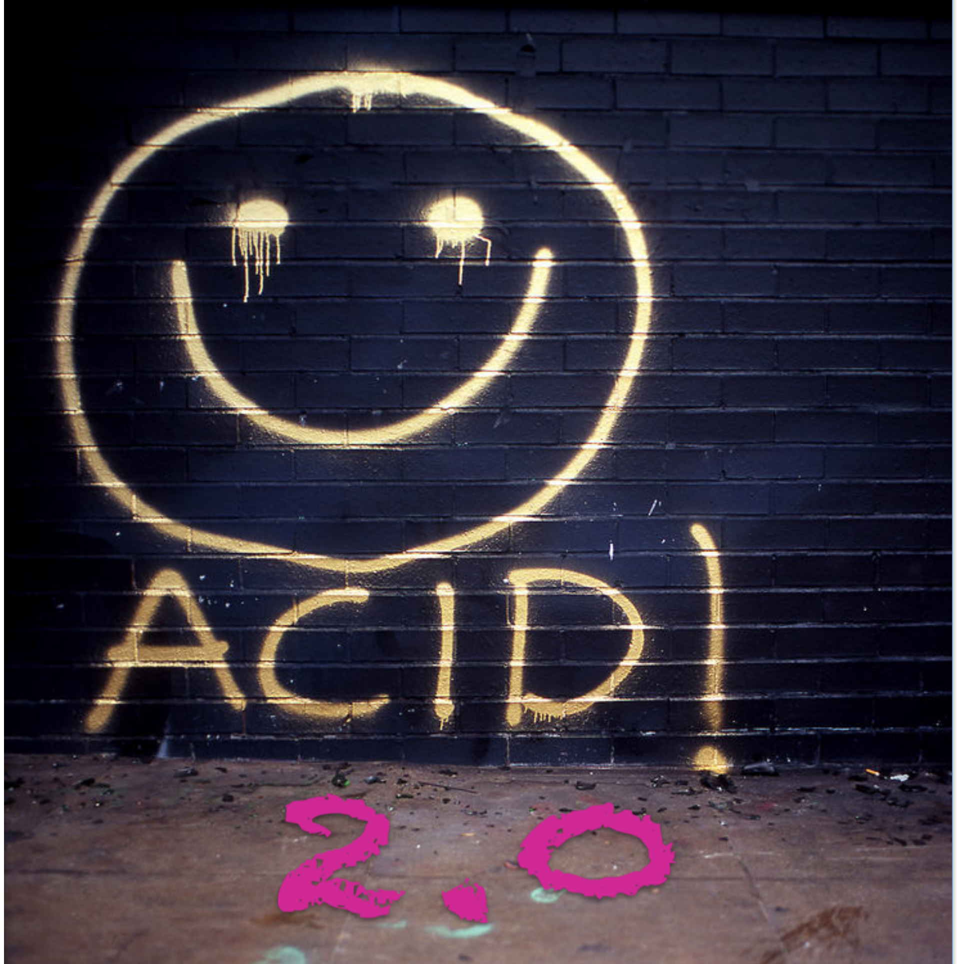
COMPILERS

TRUMP

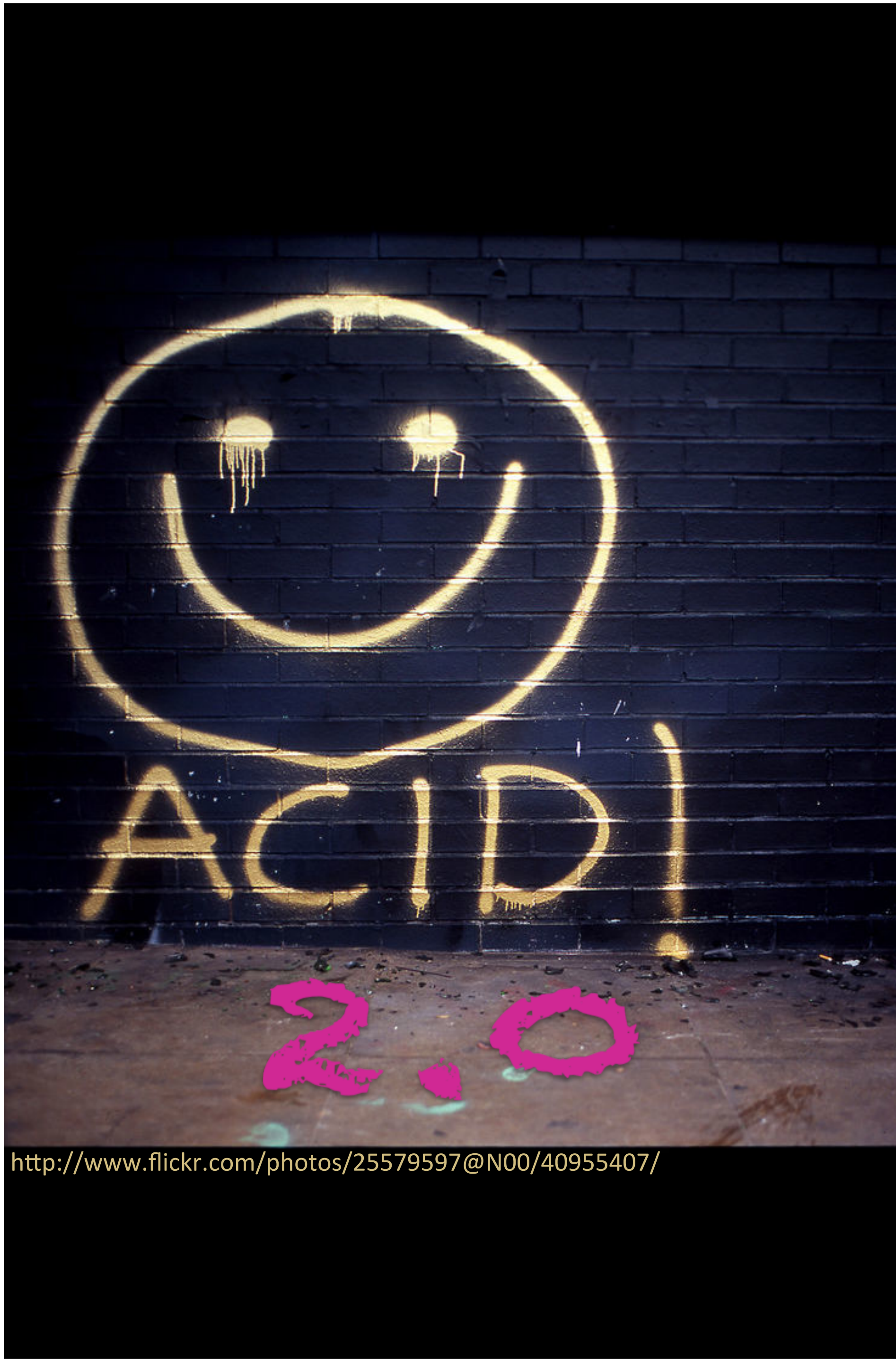
INFRASTRUCTURE

ELEGANCE AND DISORDER

- ELEGANT THEORY
  - Maxims ⇒ Theorems
    - Lattices
    - CALM Theorem
- PRACTICAL ENGINEERING
  - Theorems ⇒ Compilers
    - `<~ bloom`
    - CALM Analysis



http://www.flickr.com/photos/25579597@N00/40955407/



Associative
- $(X \circ Y) \circ Z = X \circ (Y \circ Z)$
- **batch-insensitive**

Commutative
- $X \circ Y = Y \circ X$
- **order-insensitive**

Idempotent
- $X \circ X = X$
- **resend-insensitive**

Distributed
- **acronym-insensitive**

Associative

- $(X \circ Y) \circ Z = X \circ (Y \circ Z)$
- batch-insensitive

Commutative

- $X \circ Y = Y \circ X$
- order-insensitive

Idempotent

- $X \circ X = X$
- resend-insensitive

Distributed

- acronym-insensitive

Storing an Integer

VON NEUMANN

```
int ctr;

operator:= (x) {
   // assign
   ctr = x;
}
```

ACID 2.0

```
int ctr;

operator<= (x) {
   // merge
   ctr = MAX(ctr, x);
}
```

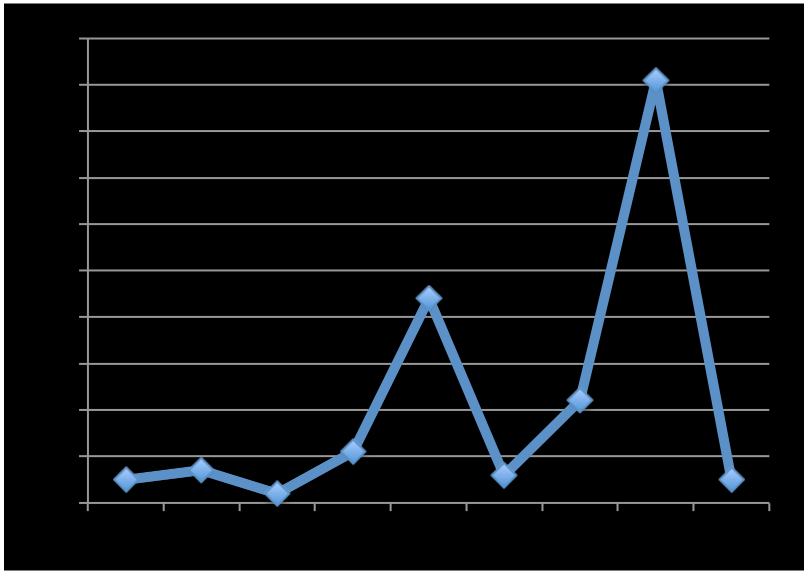
DISORDERLY INPUT STREAMS:

2, 5, 6, 7, 11, 22, 44, 91

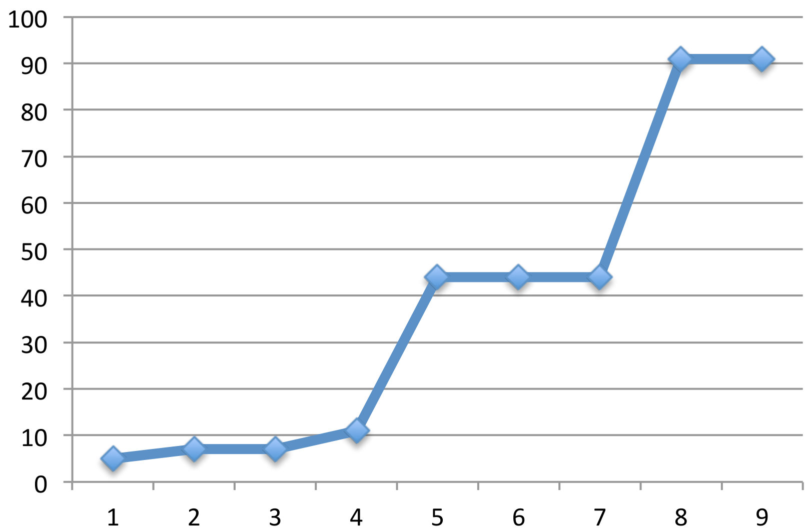
5, 7, 2, 11, 44, 6, 22, 91, 5

Storing an Integer

VON NEUMANN



ACID 2.0



DISORDERED INPUT STREAMS:

2, 5, 6, 7, 11, 22, 44, 91

5, 7, 2, 11, 44, 6, 22, 91, 5

PROGRESS

- **Lemma:**
  - ACID 2.0 ⇒ *monotonic*
- **Lemma:**
  - ACID 2.0 ⇒ *confluent*
- **Corollary:**
  - ACID 2.0 ⇒ *convergent*
  - a.k.a. "Eventually Consistent"
    - No coordination!

CRDTs

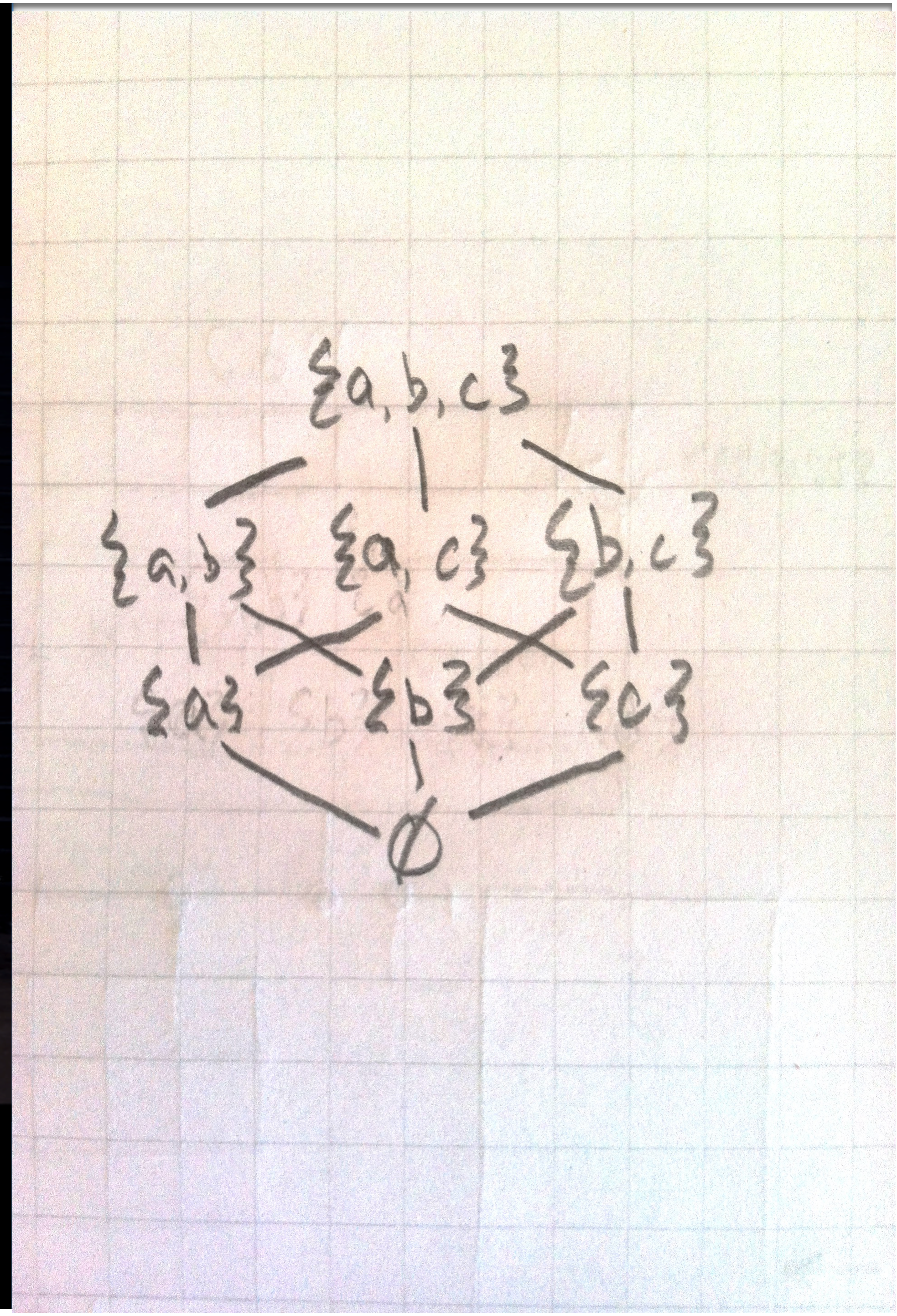
convergent replicated data types
[Shapiro, et al. 2011]

- Semilattice objects
  - A class
  - merge() is ACID 2.0
- Many examples:
  - int w/ Max
  - set w/ Union
  - map w/Insert
  - …

CRDTs

convergent replicated data types

[Shapiro, et al. 2011]



CRDTs

convergent replicated data types

[Shapiro, et al. 2011]

SCOPE DILEMMA

- SINGLE-OBJECT PROGRAMS?
- PROVE ACID 2.0
  - formalism?
  - unit testing?

DESIRE: COMPOSITION

- PIECEWISE ANALYSIS
  - Multiple simple CRDTs
  - Each easy to test
  - Rules for composition
- SET LATTICES KNOWN
  - Database query languages
    - select/project/join rules
    - even with recursion!
  - Distributed Datalog
    - see P2, etc.
- CONSISTENCY?

THE
CALM
THEOREM

Consistency As Logical Monotonicity

THE CALM THEOREM

1. MONOTONICITY
   ⇒ EVENTUAL
   CONSISTENCY

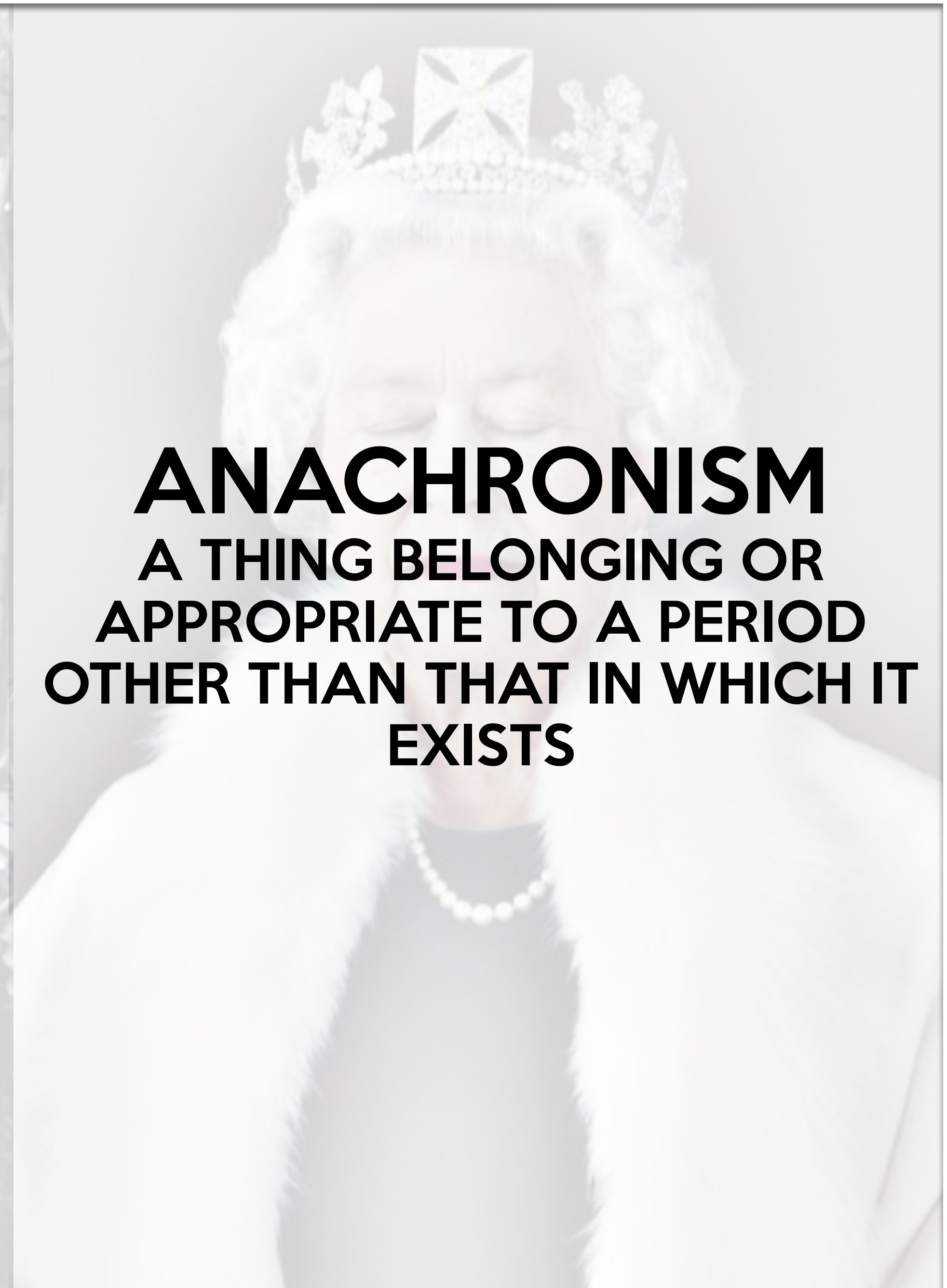
2. WHEN TO
   COORDINATE?
   NON-MONOTONE
   OPERATORS

KEEPING
CALM

THEORY
$\Rightarrow$ COMPILER

COMPILERS
TRUMP
INFRASTRUCTURE

http://www.flickr.com/photos/scobleizer/4870003098/sizes/l/in/photostream/



ANACHRONISM

A THING BELONGING OR APPROPRIATE TO A PERIOD OTHER THAN THAT IN WHICH IT EXISTS

"anachronism". Oxford Dictionaries. April 2010. Oxford Dictionaries. April 2010. Oxford University Press. 07 October 2012 <http://oxforddictionaries.com/definition/english/anachronism>.

A *disorderly* language of lattices and mappings.

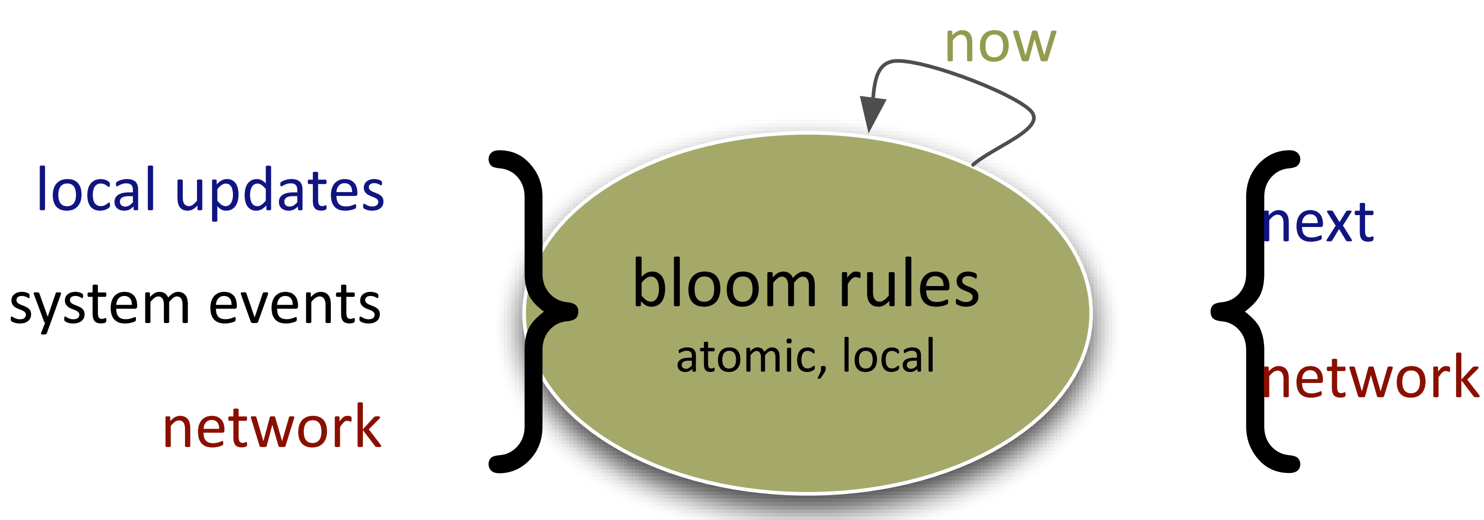
Encourages monotonicity.

Highlights non-monotonicity.

Designed for distribution.

<~ bloom operational model

- really a metaphor for a logic called dedalus
- each node runs independently
  - local clock, local data, local execution
- timestepped execution loop at each node



Hello World in <~ bloom

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
end
```

Hello World in <~ bloom

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast    <~ (mcast*nodelist).pairs { |m,n|
```

sets of key/value pairs

Hello World in <~ bloom

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast    <~ (mcast*nodelist).pairs { |m,n|
```

instantaneous merge (union)

async merge (union)

Hello World in <~ bloom

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast    <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
end
```

monotone functions

Monotone Function

f **monotone if**

$$x \leq y \Rightarrow f(x) \leq f(y)$$

Hello World in <~ bloom

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast    <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
end
```

monotone functions

Tables and Channels

```
state do
  table   :nodelist
  channel :connect
  channel :mcast


end
```

See Getting Started docs on github

```
# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }


end
```

Hello World in <~ bloom

```
state do
  table   :nodelist
  channel :connect
  channel :mcast


end

# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }


end
```

Lattice merge

+

Monotone Functions

MONOTONIC PROGRAM

hence

EVENTUALLY CONSISTENT

More Lattices

```
state do
  table   :nodelist
  channel :connect
  channel :mcast
  lmax    :cnt
  lbool   :crowded
end

# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
  cnt <= nodelist.group([], count())
  crowded <= cnt.gt(100)
end
```

More Lattices

```
state do
  table   :nodelist
  channel :connect
  channel :mcast
  lmax    :cnt      # integer with MAX
  lbool   :crowded  # bool with OR
end

# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
  cnt <= nodelist.group([], count())
  crowded <= cnt.gt(100)
end
```

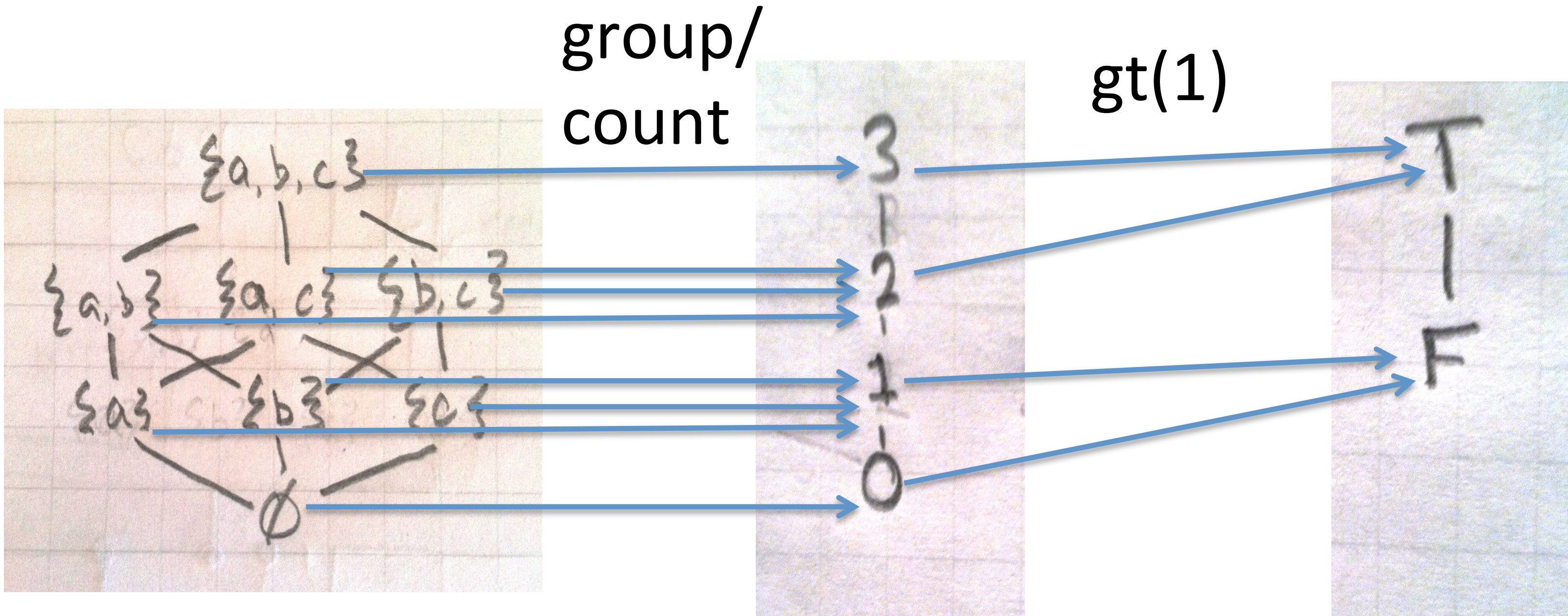
More Lattices

```
state do
  table   :nodelist
  channel :connect
  channel :mcast
  lmax    :cnt      # integer with MAX
  lbool   :crowded  # bool with OR
end

# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
  cnt <= nodelist.group([], count())
  crowded <= cnt.gt(100)
end
```

monotone functions
across lattice types

Monotone Functions Across Lattice Types



Non-Monotonicity Downstream of Asynchrony

```
state do
  table   :nodelist
  channel :connect
  channel :mcast
  channel :disconnect

end

# a chat server
bloom do
  nodelist <= connect.map {|c| c.val}
  mcast <~ (mcast*nodelist).pairs { |m,n|
    [n.key, m.val]
  }
  nodelist <- disconnect.map {|c| c.val}

end
```

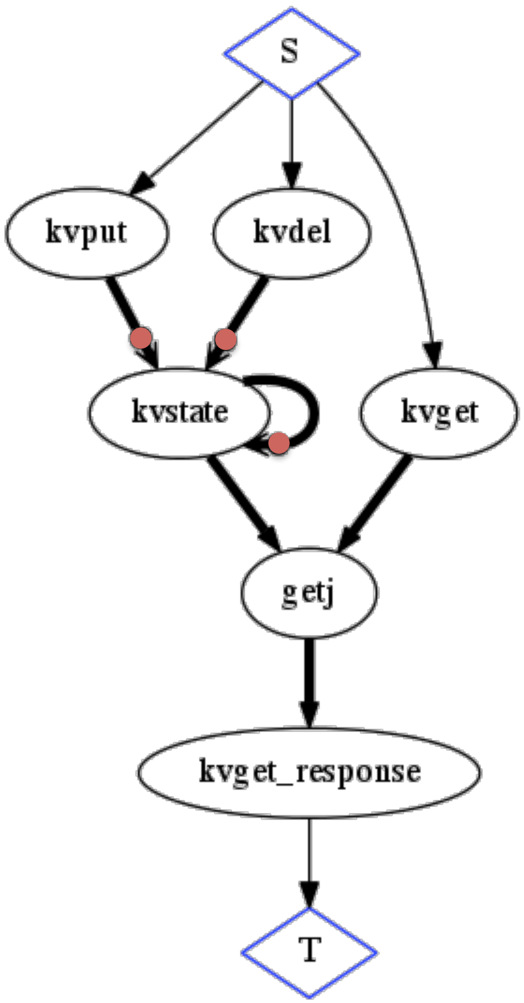
async collection

Non-monotone function

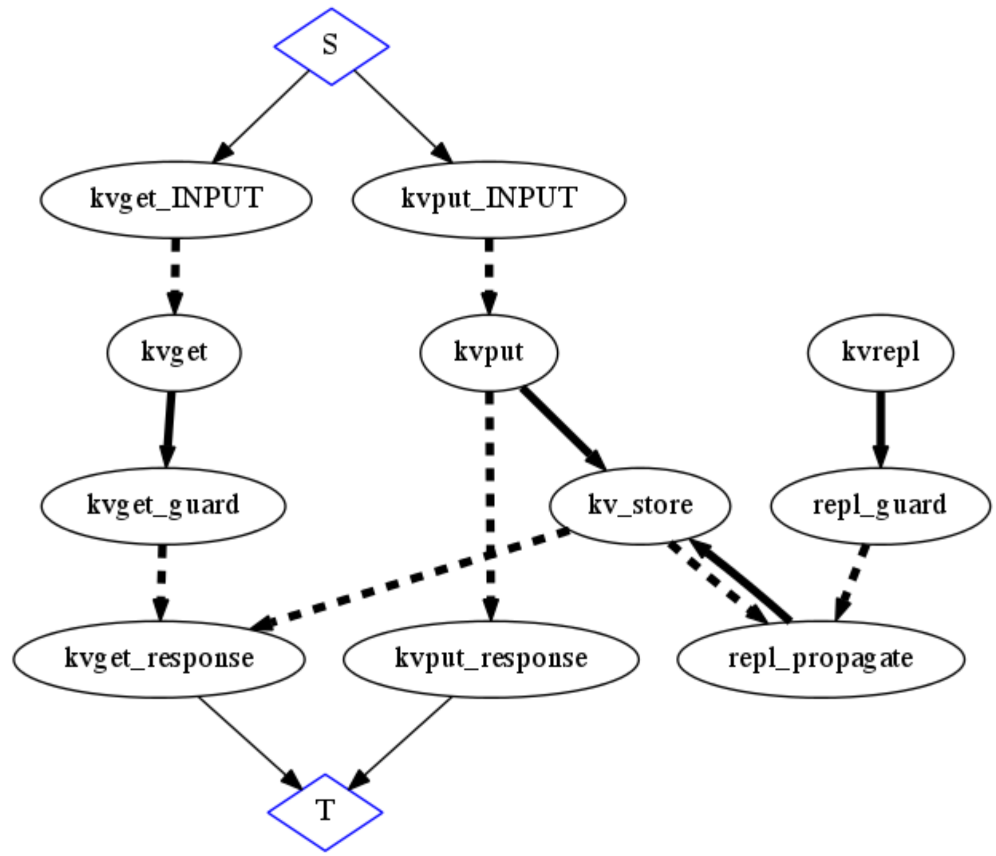
CALM Analysis

1. For any path through a Bloom module, label:
   - Asynchrony
   - Non-Monotonicity
   - Inconsistency
2. Compute labels transitively across modules
3. Identify code that needs coordination
4. Assess comm pattern, suggest coordination
   - 1-1, 1-many : ordered delivery
   - many-many : order proxy, Paxos, etc.

Alvaro Diagrams



Basic KVS



Lattice KVS

Vector Clocks in Bloom

```
module VectorClock
  state do
    lmap :my_vc
    lmap :next_vc
    scratch :in_msg, [:addr, :payload] => [:clock]
    scratch :out_msg, [:addr, :payload]
    scratch :out_msg_vc, [:addr, :payload] => [:clock]
  end

  bootstrap do
    my_vc <=      {ip_port => Bud::MaxLattice.new(0)}
  end

  bloom do
    next_vc <= out_msg { {ip_port => my_vc.at(ip_port) + 1} }
    out_msg_vc <= out_msg {|m| [m.addr, m.payload, next_vc]}
    next_vc <= in_msg  { {ip_port => my_vc.at(ip_port) + 1} }
    next_vc <= my_vc
    next_vc <= in_msg  {|m| m.clock}
    my_vc <+ next_vc
  end
end
```

Vector Clocks
bloom v. wikipedia

```
bootstrap do
  my_vc <=
     {ip_port => Bud::MaxLattice.new(0)}
end

bloom do
  next_vc <= out_msg
    { {ip_port => my_vc.at(ip_port) + 1} }
  out_msg_vc <= out_msg
    {|m| [m.addr, m.payload, next_vc]}
  next_vc <= in_msg
    { {ip_port => my_vc.at(ip_port) + 1} }
  next_vc <= my_vc
  next_vc <= in_msg  {|m| m.clock}
  my_vc <+ next_vc
end
```

Initially all clocks are zero.

- Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
- Each time a process prepares to send a message, it increments its own logical clock in the vector by one and then sends its entire vector along with the message being sent.

Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

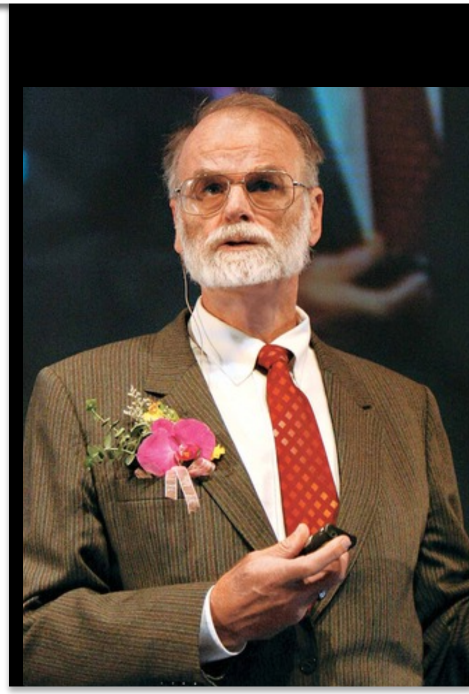
THE
CALM
THEOREM

COROLLARIES:
WHY COORDINATE?

CRON
CAUSALITY REQUIRED ONLY FOR NON-MONOTONICITY

COORDINATION COMPLEXITY
HOW MUCH COORDINATION IS TRULY NEEDED FOR YOUR ALGORITHM?

FATEFUL TIME
THE ONLY USE FOR “TIME” IS TO “SEAL FATE”.



SUMMARY:
UNITY

MAXIMIZE DISORDER
AND
UNDERSTAND ORDER'S ROLE

DISORDERLY CODE
AND
WHOLE-PROGRAM ANALYSIS

BOOM TEAM

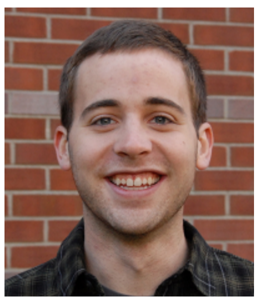

joe hellerstein


david maier


ras bodik


alan fekete


peter alvaro

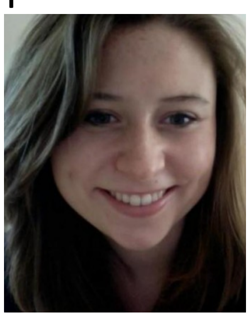

peter bailis


neil conway


bill marczak


haryadi gunawi


sriram srinivasan


emily andrews

andy hutchinson

bloom-lang.org
boom.cs.berkeley.edu

- Papers
  - CALM/Bloom, CIDR '11
  - Bloom+Lattices, SOCC '12
  - BloomUnit, DBTest '12
- Videos
  - Declarative Imperative, PODS '10
  - Bloom, Lang.Next '12
  - Bloom+Lattices, Basho Meetup '12

ACM SOCC NEXT TUES 10/16
10:45AM
SAN JOSE MARRIOT

DATA CONSISTENCY SESSION

- NEIL CONWAY ON LATTICE SUPPORT IN BLOOM
- PETER BAILIS ON POTENTIAL DANGERS OF CAUSAL CONSISTENCY

http://www.socc2012.org

KEEP
CALM
AND
QUERY
ON

POSITIVE THINKING
FOR THE
CLOUDY FUTURE

bloom-lang.org