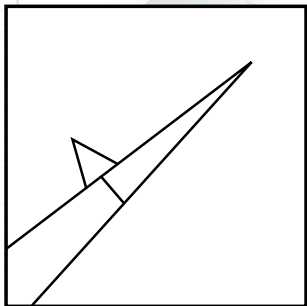




**Martin is getting the  
projector  
to work  
with his  
laptop.**

# Beyond Threads

---



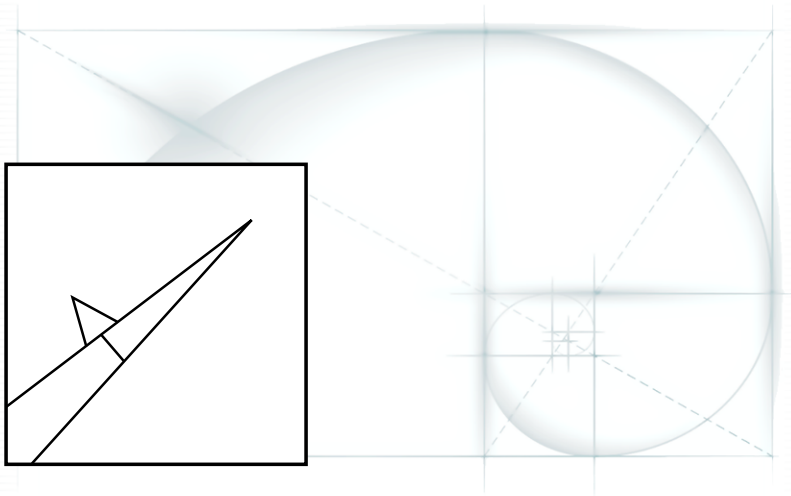
Concurrency,  
E,  
and Smalltalk

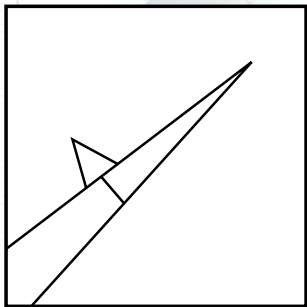
---

Martin McClure

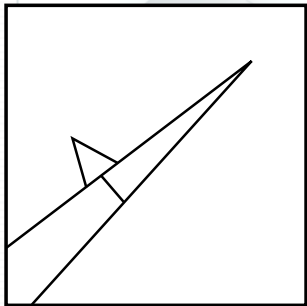


GEMTALK™  
SYSTEMS





**A unicorn  
at a  
birthday party**



**A unicorn  
at a  
birthday party**

**(Thanks, Nancy!)**

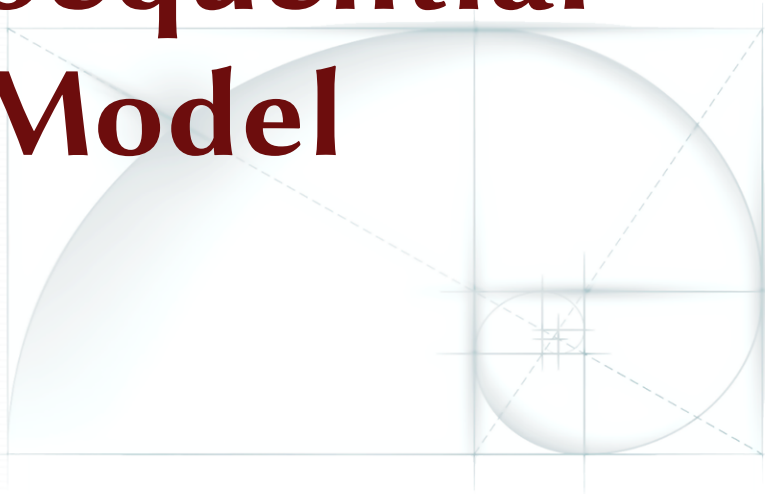
# Beyond Threads



# Threads



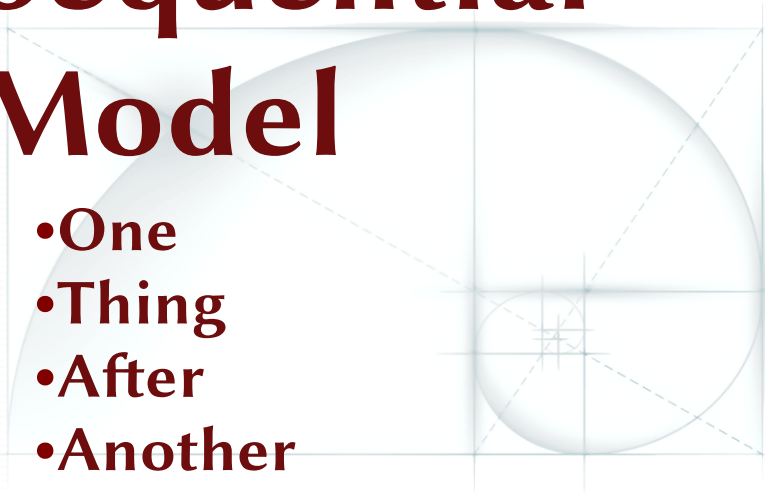
# Sequential Model





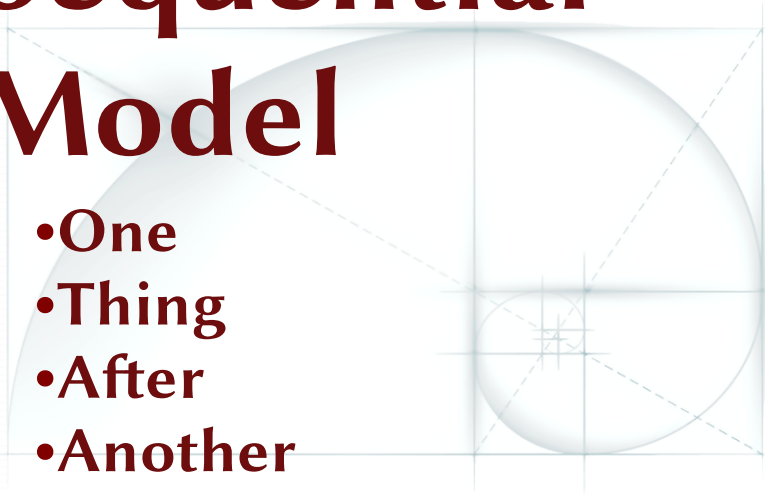
# Sequential Model

- One
- Thing
- After
- Another



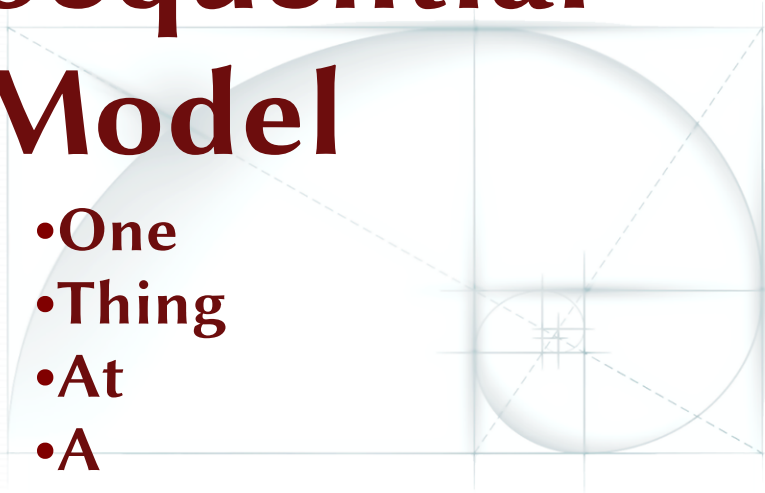
# Sequential Model

- One
- Thing
- After
- Another
- (branching, call/return)



# Sequential Model

- One
- Thing
- At
- A
- Time



# Shared-State Thread Model

- Many threads
- Each is sequential
- State is shared

# Thread

## Advantages

- >1 thing “at once”
- Program sequentially

# Thread

# Drawbacks

- Program sequentially
- Execute randomly

# Thread Drawbacks



- Race conditions

# Thread Drawbacks



- Race conditions
- Deadlock



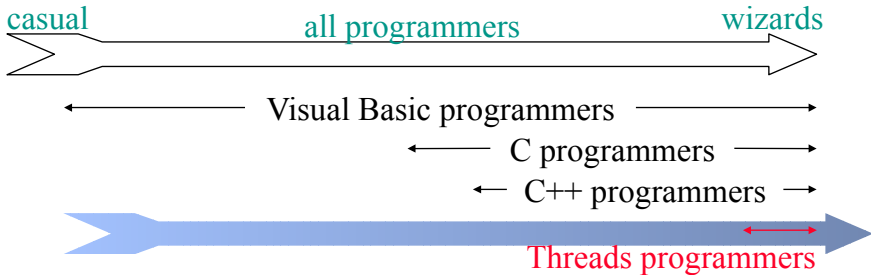
# Thread Drawbacks

- **Must understand entire system. Impossible.**



**Threads  
are  
Bad**

# What's Wrong With Threads?



**Too hard for most programmers to use.**

**Even for experts, development is painful.**

# **“The Problem with Threads”**

**Edward A. Lee**

# Beyond Threads



# Beyond Threads

A golden rectangle is shown with a golden spiral inside it. The spiral starts from the center and winds outwards, touching the sides of the rectangle. A large, bold, dark red letter 'E' is positioned in the lower-left quadrant of the rectangle. The entire graphic is overlaid on a light blue grid.

**E**



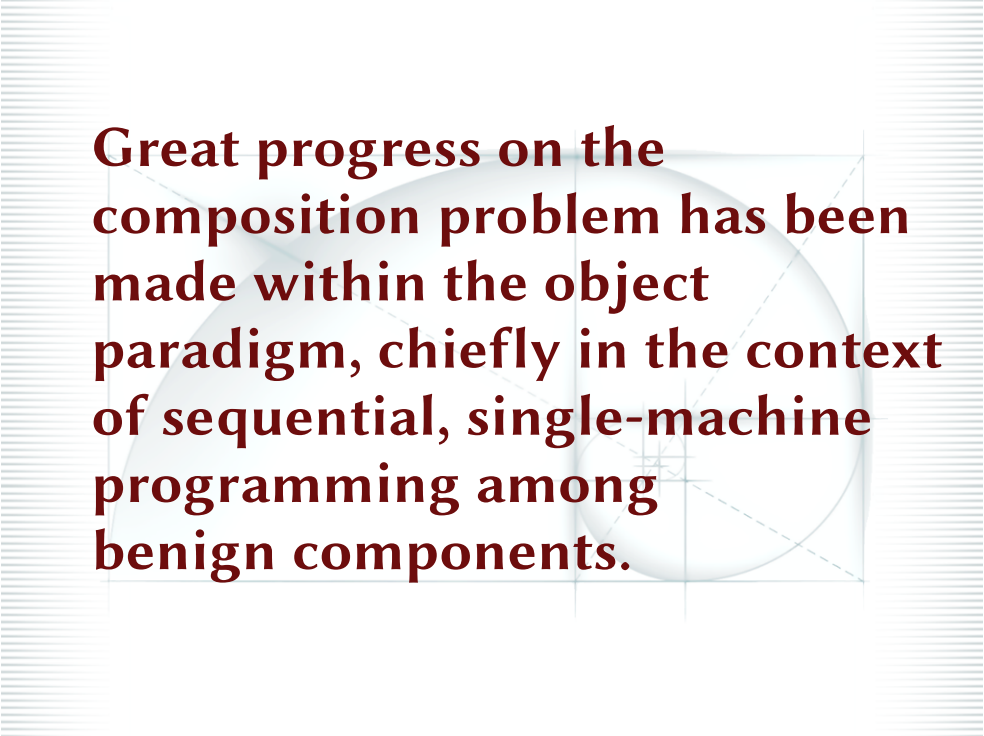
**These ideas are  
not new.**



**What's  
E  
all about?**



**When separately written programs are composed so that they may cooperate, they may instead destructively interfere in unanticipated ways. These hazards limit the scale and functionality of the software systems we can successfully compose.**

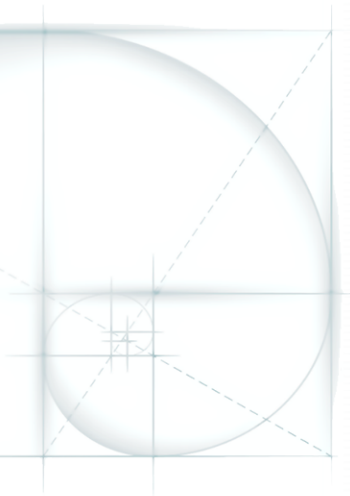


**Great progress on the composition problem has been made within the object paradigm, chiefly in the context of sequential, single-machine programming among benign components.**

**We ... extend this success to support robust composition of concurrent and potentially malicious components distributed over potentially malicious machines.**

# E

- **Objects**
- **Composition**
- **Concurrency**
- **Distribution**
- **Security**



A technical drawing of a dome or semi-sphere. The dome is shaded in a light blue-grey gradient. It is overlaid with a grid of thin grey lines. A golden spiral is drawn on the right side of the dome, starting from a central point and spiraling outwards. Dashed lines represent the dome's profile and the spiral's path. The word "Vat" is written in a bold, dark red serif font across the center of the dome.

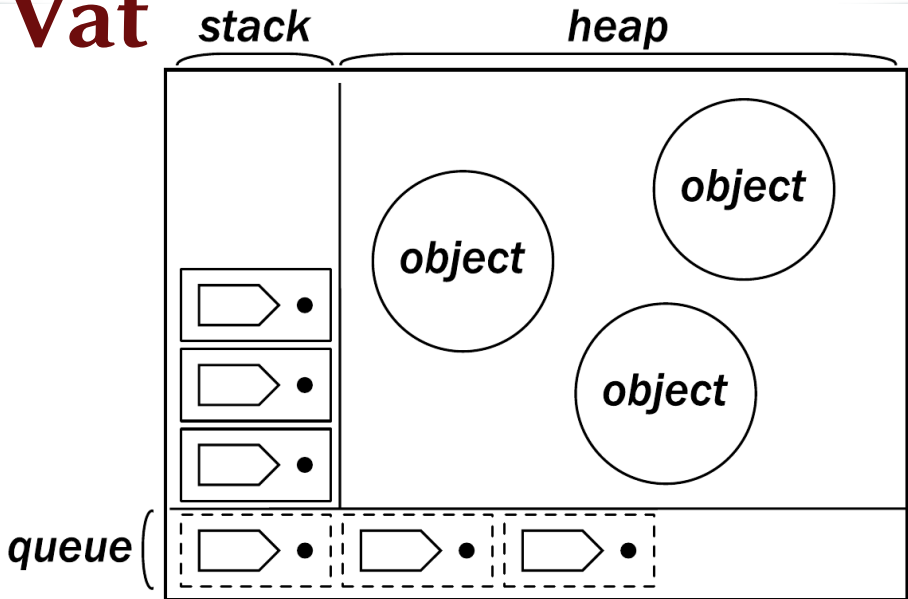
**Vat**

# Two kinds of send

- immediate
- eventual



# Vat



# Two kinds of object reference

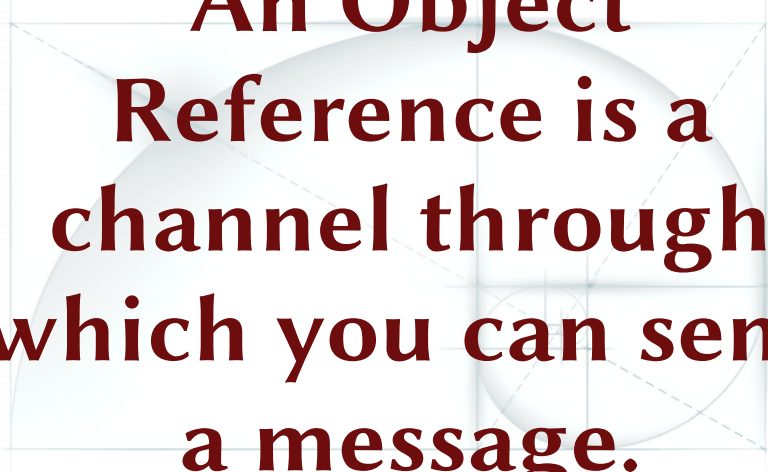




# Object Reference



- **Assign to a variable**
- **Send a message**



**An Object  
Reference is a  
channel through  
which you can send  
a message.**

# Two kinds of object reference

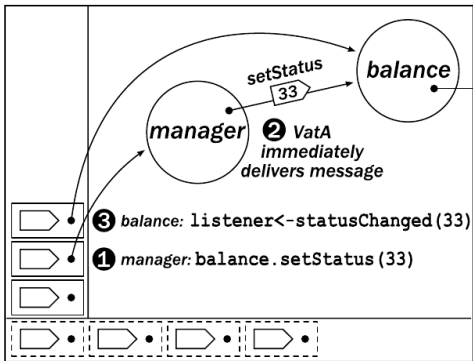
- near
- eventual



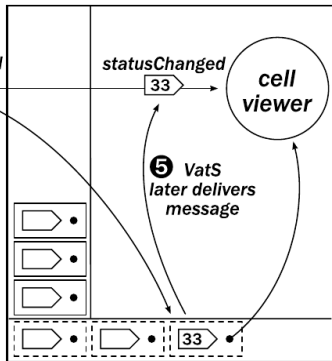
# Two kinds of object reference

- **near**
  - immediate send
  - eventual send
- **eventual**
  - eventual send only

## account vat (VatA)



## spreadsheet vat (VatS)



# Message

- receiver
- selector
- arguments



# Arguments

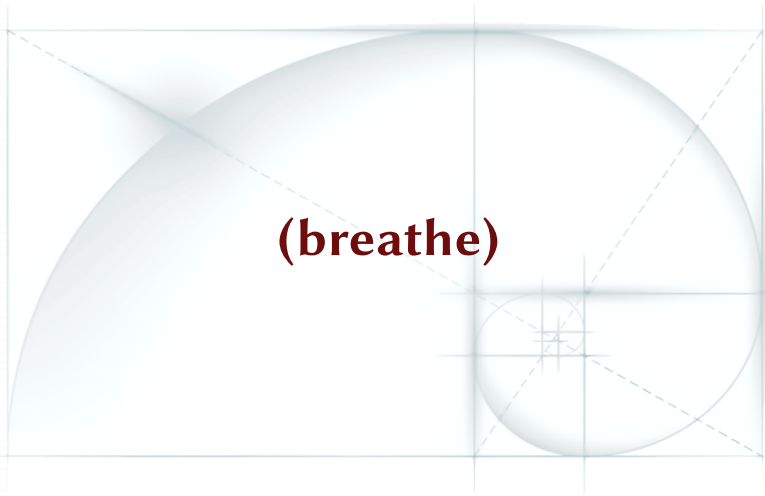
- **Near references in the sending vat become eventual references when the message is received in another vat.**

# Data Objects

- **Transitively immutable**
- **Passed by copying**
- **Receiver sees a near reference in its vat**



**(breathe)**





**Immediate send  
waits for a response  
before returning**

**Eventual send  
returns immediately**

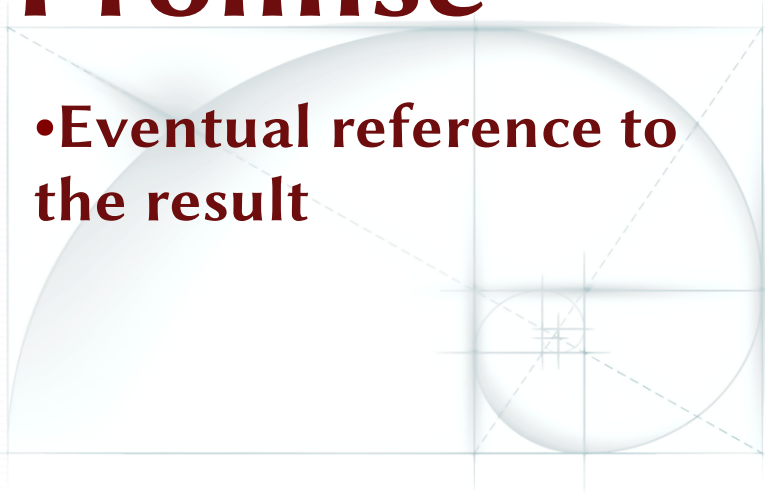
A faint, light blue background graphic featuring a Fibonacci spiral (golden spiral) overlaid on a grid. The spiral starts from a central point and winds outwards, passing through several squares of decreasing size. The grid consists of solid lines forming a rectangle, with dashed lines extending from the corners to the center of the spiral.

**Eventual send  
returns immediately**

**But what does it  
return?**

# Promise

- **Eventual reference to the result**



# Promise

- **Resolver**
  - **One for each promise**
  - **Sent with message**
  - **Tells promise what object it represents**

# Promise

- Messages sent to promise
  - Before resolved
  - Queued
- When resolved
  - Queued messages sent
- Once resolved
  - Equivalent to resolution

# Latency




```
workQueue removeFirst  
process = 'done'.
```



# Pipelining



```
workQueue removeFirst  
process = 'done'.
```



**When Things  
go  
Wrong**

# Exceptions

- Promise resolves to broken reference
- Any message sent to the promise signals the exception in the sending vat

# Odds & Ends

A large, faint, light blue Fibonacci spiral is centered in the background of the slide. It is overlaid on a light gray grid. The spiral starts from the bottom right and moves counter-clockwise, passing through the top right, top left, and bottom left corners of the grid.

- **When-Catch expression**
  - **Multi-way join**
- **Guaranteed order of delivery**



**Is all this a  
good idea?**

# Advantages

- No race conditions
- No deadlock
- Fairly straightforward model
- Enables distribution
- More easily enables multi-core use

# Drawbacks



- **Datalock**
- **Multi-vat recursive algorithms require special handling**



**Is all this a  
good idea?**





**What would  
it take?**

# Changes



- **Syntax for eventual send**
- **Better support for data objects**



**# = = = #**

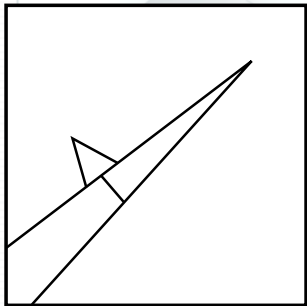


**E**

**erights.org**

# Beyond Threads

---



Concurrency,  
E,  
and Smalltalk

---

Martin McClure



GEMTALK™  
SYSTEMS