CORNELIA DAVIS
SR. DIRECTOR OF TECHNOLOGY
PIVOTAL

RETHINKING THINKING MODELS FOR EVENT-DRIVEN PROGRAMMING
function SumToN(n : INTEGER): INTEGER;
var
  i : INTEGER;
  s : INTEGER;
begin
  i := 0;
  s := 0;
  while (i <= n) do
    begin
      s := s + i;
      i := i + 1;
    end;
  SumToN := s;
end;
(define sumToN
  (lambda (n)
    (if (= n 1)
        1
        (+ n (sumToN (- n 1)))
  )))
function SumToN(n : INTEGER): INTEGER;
var
  i : INTEGER;
  s : INTEGER;
begin
  i := 0;
  s := 0;
  while (i <= n) do
  begin
    s := s + i;
    i := i + 1;
  end;
  SumToN := s;
end;
(define sumToN
  (lambda (n)
    (if (= n 1)
      1
      (+ n (sumToN (- n 1)))))
)

(BASE CASE) (CONDITIONAL)

(GENERAL CASE)
(INCLUDES A LEAP OF FAITH)
function SumToN(n : INTEGER): INTEGER;
var
  i : INTEGER;
  s : INTEGER;
begin
  i := 0;
  s := 0;
  while (i <= n) do
    begin
      s := s + i;
      i := i + 1;
    end;
  SumToN := s;
end;

(define sumToN
  (lambda (n)
    (if (= n 1)
      1
      (+ n (sumToN (- n 1)))))
  )

RECURSION IS HARD!!

RECURSION IS NATURAL
WHAT ARE THE PRIMITIVES, PATTERNS AND PLATFORMS THAT WILL MAKE EVENT-DRIVEN PROGRAMMING EASY?
FUNCTIONS AND EVENTS?
MAYBE

I HONESTLY HAVEN’T GOT A CLUE
THE "IMPERATIVE" MODEL?

- Cloud-Native App  
  (yup, the microservices)

- Cloud-Native Services  
  (yup, some are apps  
  ... and other are other types  
  of services)

- Cloud-Native Data  
  (a distributed data fabric)

- The Cloud-Native Collective  
  (implicit and explicit connections)
Microservices - Not A Free Lunch!

TUESDAY, APRIL 8, 2014 AT 8:54AM

This is a guest post by Benjamin Wootton, CTO of Contino, a London based consultancy specialising in applying DevOps and Continuous Delivery to software delivery projects.

Microservices are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services.

WE’VE BEEN THINKING CLIENT/SERVER FOR SO LONG

- HA & Scaling
  - → Multiple Instances
  - → Load Balancing
  - → Externalize Config

- Resilience
  - → Statelessness
  - → Retries
  - → Circuit Breaker
**Autonomy and Agility**
- → Versioning

**Dynamic Environment**
- → Service Discovery
- → Resource Bindings
IN SERVICE TO “SERVICES”

- Agility and Autonomy
  - Services get their own DB
    - MDM? (ehh)
- Resilience
  - Caches
    - Cache miss
  - Expiry
WE’VE BEEN THINKING CLIENT/SERVER FOR SO LONG

- **Managability**
  - → Distributed Tracing
- Act locally - who’s thinking globally?
DISTRIBUTED SYSTEMS CONCERNS

- HA & Scaling
- Dynamic Environment
- Resilience
- Autonomy and Agility
- Managability
RESILIENCE

FROM RETRIES...

CLIENT

RETRY

SERVICE

... TO PROMISES

CLIENT

PROMISE

EVENT

GENERAL CASE
(INCLUDES A LEAP OF FAITH)

FROM CIRCUIT BREAKERS...

... ENSURING EVENTS ARE NOT LOST

@cdavisafc
FROM CACHE EXPIRY... 

SERVICE

CACHE EXPIRY

GENERAL CASE
(INCLUDES A LEAP OF FAITH)

VIEW

SERVICE

EVENT CONSUMER

... ENSURING EVENTS ARE NOT LOST

... TO MATERIALIZED VIEWS
RESILIENCE & SCALE

FROM SERVICES...  ... TO FUNCTIONS

FROM LOAD BALANCING...  ... TO FUNCTIONS

A BIT LIKE THE STACK FRAME WITH EACH RECURSIVE CALL
FROM DATA BOTTLENECK . . .

. . . DATA PARTITIONING

. . . AND PARTITION RESOLVERS
FROM SHARED DATABASES...

EVENT SOURCING
ALSO, EVENT PRODUCERS
MANY QUESTIONS

- Programming the “Collective”
- Justifying the “leap of faith”
- ...

@cdavisafc
CLOUD-NATIVE - IMPERATIVE MODEL

- Cloud-native Apps
  - Scale out
  - Statelessness
  - Externalize configuration
  - Implications from changes in the application lifecycle
  - Retries

- Services
  - Versioned services (autonomy)
  - Service Discovery
  - Distributed tracing

- Data
  - Breaking the Data monolith
  - Data APIs
  - Caching
  - Polyglot Persistence
  - Event sourcing
CLOUD-NATIVE - EVENT-DRIVEN MODEL

- Promises
- Event Consumers
- Event Producers
- Distributed Data Fabric: Data Partitioning and Partition Resolvers
- Event Sourcing and Materialized Views
- Health check (are producers & consumers functioning?)
Hard to explain in layman terms in just a sentence. I’ll try to make it as layman as possible.

Suppose you have a function f which takes one argument which is supposed to be a continuation. Then call/cc is a special function that takes f as an argument and calls f passing it, as its argument, the current continuation, which is the continuation in which call/cc itself was called.
THANK YOU