Event-Driven Execution

CS520
Dept. of Computer Science
Univ. of New Hampshire

Program is thought of as responding to a series of events:

Events could be external:
- mouse or keyboard event
- I/O operation completion

Events could be internal:
- e.g. used to introduce abstractions to simplify concurrency

Event Handlers

Functions are designated as handlers for specific events

System will call handler for event when it occurs

Event Loop

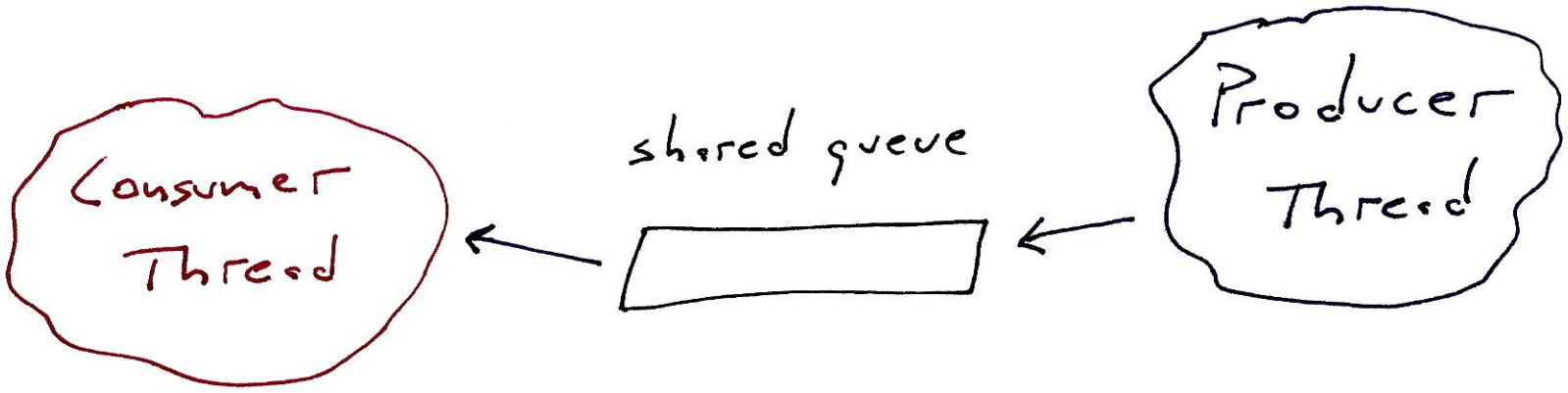
system is running an event loop
at the core of the program

when event occurs, handler function
is enqueued for execution by
the event loop

program terminates when no handler
is executing, the queue is empty,
and no more events can occur

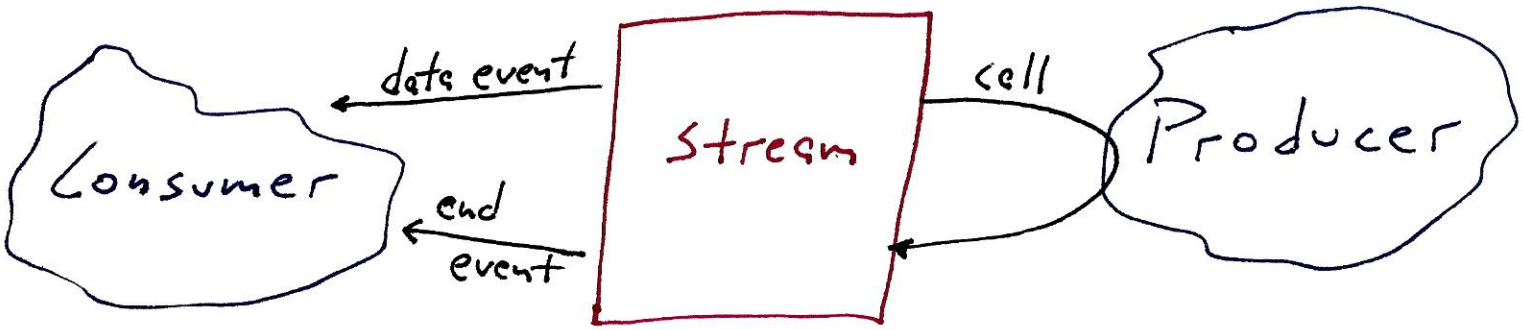
the program initiation event has a
handler — the main function

Producer – Consumer Pattern for Threads



interaction is coordinated explicity by use of mutexes and condition variables

Event-Driven Stream



Stream is an abstraction:

- contains hidden internal queue
- provides concurrency
  - e.g. Producer is run in a separate thread from the event loop

User View

Create a Stream and override its "produce" method

define a handler for the Stream's "data" event

define a handler for the Stream's "end" event

start the Stream

Stream View

create an internal queue

create a thread to:

call the "produce" method

insert produced value into queue

announce a "data" event

thread repeats these three steps

until "produce" method says "done"

thread is blocked when queue is full

an "end" event is announced when "produce"

method says "done"

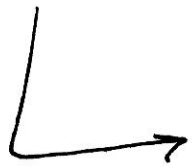
System View

provide an event loop

queue of functions to be called

when event occurs, its handler is placed in queue

thread repeatedly pulling next function from queue and executing it



thread needs to be blocked if queue is empty

javascript in browsers

↳ based upon event loop

Node.js — server side javascript