Software Architecture

Lecture 7
Communicating Peers

João Pedro Sousa
George Mason University

previously, event systems within the interacting processes family

data flow
- batch sequential
- dataflow network (pipe & filter)
  - acyclic, fan-out, pipeline, Unix
- closed loop control

call-and-return
- main program/subroutines
- information hiding – objects
- stateless client-server
- SOA

interacting processes
- communicating processes
- event systems
  - implicit invocation
  - publish-subscribe

data-oriented repository
- transactional databases
  - stateful client-server
- blackboard
- modern compiler

data-sharing
- compound documents
- hypertext
- Fortran COMMON
- LW processes

hierarchical
- tiers
  - interpreter
  - N-tiered client-server

remember: communication is loosely coupled in the interacting processes family

- components
  - independent threads of control
  - implemented as a process or thread
  - may be distributed
- connectors
  - communication is asynchronous and loosely coupled
- system
  - components may or may not have knowledge of other components
  - functionality of one component does not depend upon others
  - overall system functionality depends upon all components functioning and communicating properly

today communicating peers

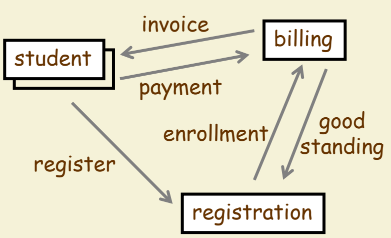
- flavors
  - homogeneous systems, aka peer-to-peer (P2P)
  - heterogeneous systems
- QAs
- understanding concurrency & distribution
  - pool vs. factory
- case study asynchronous messaging
  - QAs

example homogeneous systems peer-to-peer (P2P)

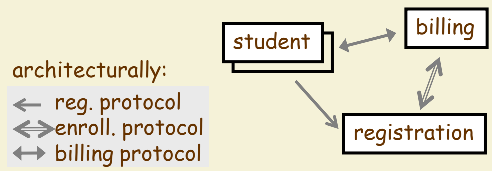
all peers play similar roles / use same protocols:

- peer-to-peer networks
  - digital telephony (VOIP)
  - internet traffic (DNS)
- mail transfer among servers (SMTP)
- discussion forums
  - Usenet news (1979)…
- file sharing protocols
  - Napster, Gnutella, BitTorrent, and dozens of others often implemented over HTTP request-reply

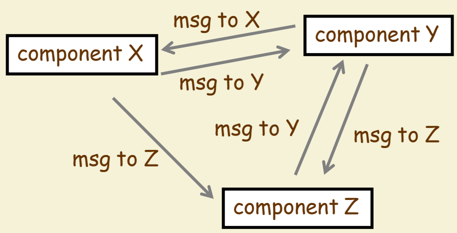
example heterogeneous system student billing system



- students register using personal devices
- registrar sends summary of enrollment to billing
- billing sends invoices
- eventually students pay
- billing informs registrar of students in good standing

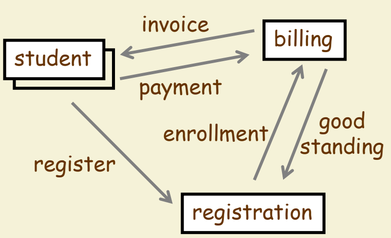


communicating peers
middle ground between call-return & events



	call-return	peers	events
identity of receiver is known	yes	yes	no
can prescribe/predict order	yes	yes	no
communication	synchronous	asynch	asynch
restrictions on topology	hierarchical	none	none

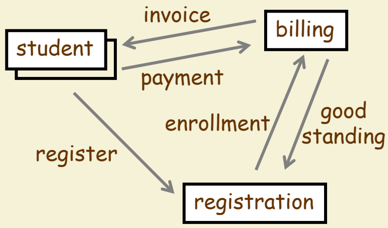
freedom of message exchange raises many questions



what if:

- a registration is received after sending the summary of enrollment
- additional enrollment entries are received after invoicing
- an invoiced payment is never received
- a payment is received after sending the list in good standing
- the list in good standing is never received

freedom of message exchange raises many questions



what if:

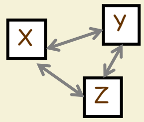
- lost messages
- duplicate messages
- unexpected messages

- protocols of interaction
  - formally specified and verified
  - ideally, designed to be robust to lost and unexpected messages
- components
  - know and follow their role in the protocols

today
communicating peers

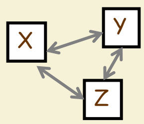
- flavors
  - homogeneous systems, aka peer-to-peer (P2P)
  - heterogeneous systems
- QAs
- understanding concurrency & distribution
  - pool vs. factory
- case study asynchronous messaging
  - QAs

communicating peers



- promote conceptual integrity
  - components work more independently than in call-return
  - interaction policy can be cleanly separated from internals
  - amenable to model and reason about concurrent behavior
- promote scalability
  - easy to add new components in homogeneous (P2P) systems
- promote responsiveness
  - asynchronous (unblocking) communication
  - concurrency (via threading) and parallelism

communicating peers



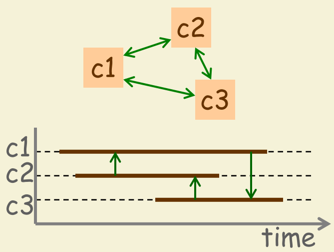
- promote robustness
  - large-scale redundancy in P2P systems
  - components and protocols are built for robustness
- promote security (relative to event systems)
  - subsets of peers can agree on encryption to keep secrets from others
- development costs may be a challenge
  - asynchronous communication and
    complexity of protocols
    adds to design, development & maintenance costs

today
communicating peers

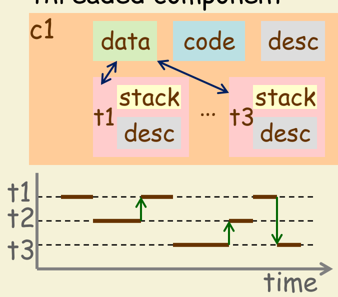
- flavors
  - homogeneous systems, aka peer-to-peer (P2P)
  - heterogeneous systems
- QAs
- understanding
  concurrency & distribution
  - pool vs. factory
- case study asynchronous messaging
  - QAs

parallelism ≠ concurrency
but similar reasoning applies

distributed
components (OS processes)



threaded component



simultaneous processing

interleaving

computations (and messages)
may occur in any order, unless explicit
steps are taken to synchronize them

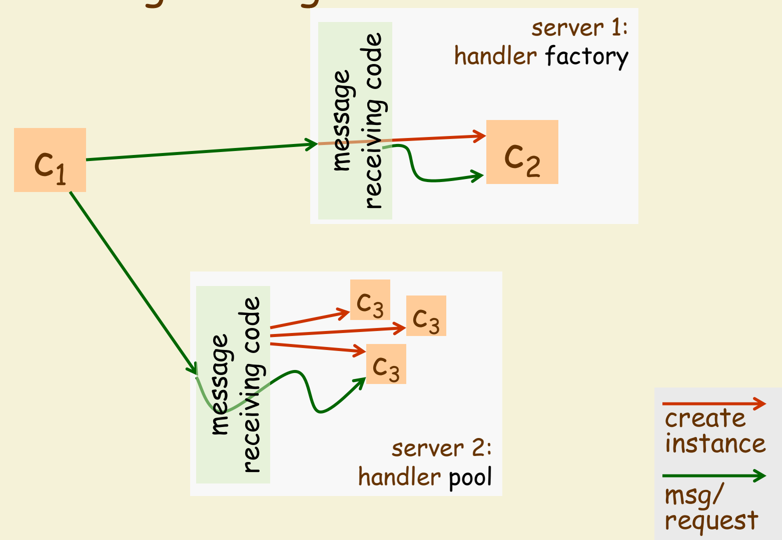
threads are supported by a library, not the OS

- why threads anyway?
  - separation of concerns:
    different activities in different threads
  - support requests of multiple peers
  - one thread remains responsive
    (e.g. handle user input or incoming messages)
    even if others are busy or blocked
    (e.g. waiting for resources, input, or messages)
- threads are supported by a library/VM, not the OS
  - making a process-blocking OS call blocks all threads
  - calling exit() in one thread terminates the process

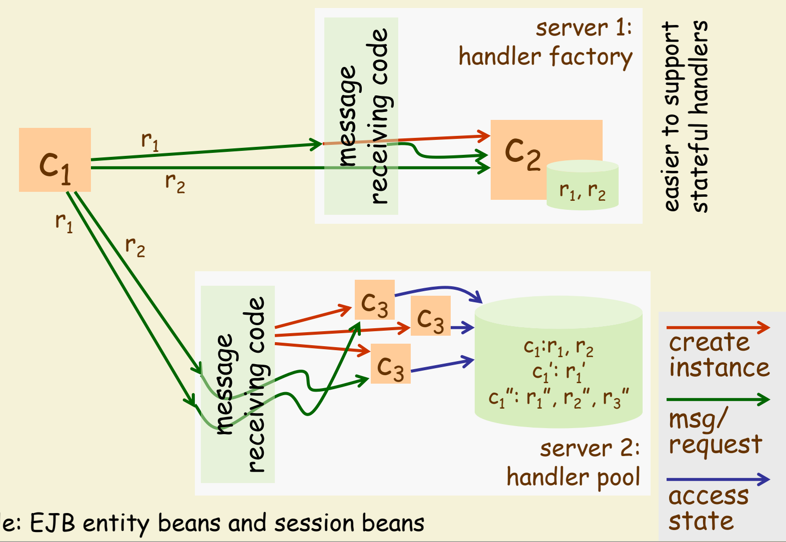
threads are often used to handle incoming messages

- in case there's a long processing associated to incoming messages
- components can be made more responsive by handling requests on separate threads
- two flavors (aka design patterns)
  - pool: assign a thread when a request comes in
    - more efficient, harder to manage
  - factory: create a thread when a request comes in
    - easier to manage, less efficient

factory vs. pool design patterns for handling messages

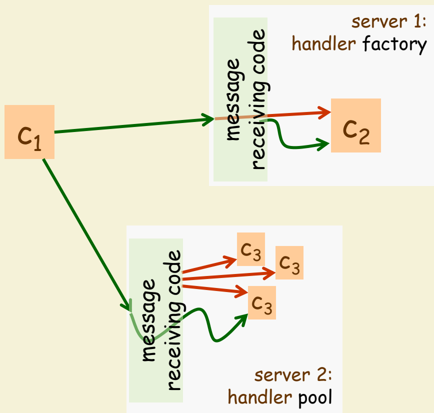


stateful handlers keep state of conversation, stateless don't



example: EJB entity beans and session beans

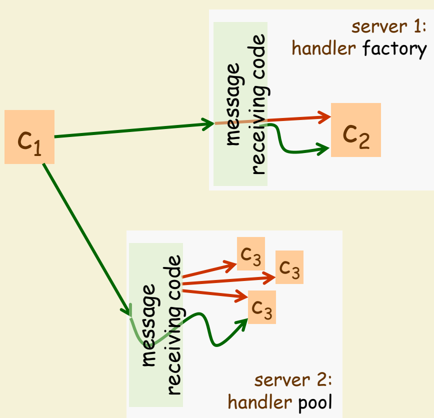
factory vs. pool design patterns promoted QAs?



- maintainability
  code complexity
- memory footprint
- response time

QA scenarios?

factory vs. pool design patterns memory footprint & response time



scenarios:

- trx avg processing is 5s
  handler creation is 1s
- load 1
  2 trx per minute
- load 2
  10 trx per minute
- load 3
  50 trx per minute
- how many replicas in the pool?

suppose load varies along the day

- load 1 during the night
- load 2 during lunch
- load 3 during business hours

take 5

today
communicating peers

- flavors
  - homogeneous systems, aka peer-to-peer (P2P)
  - heterogeneous systems
- QAs
- understanding concurrency & distribution
  - pool vs. factory
- case study asynchronous messaging
  - QAs

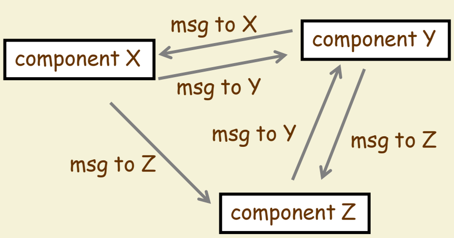
PtoP example: code view

PtoP example: run-time view

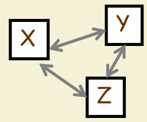
PtoP example: discussion

- which pattern does PtoP use to handle incoming messages
  - pool
  - factory
  - other?
- which QAs is PtoP promoting?

in summary, communicating peers
middle ground between call-return & events



	call-return	peers	events
identity of receiver is known	yes	yes	no
can prescribe/predict order	yes	yes	no
communication	synchronous	asynch	asynch
restrictions on topology	hierarchical	none	none



in summary, peer systems responsive & robust but costly

- QAs promoted
  - conceptual integrity
  - responsiveness
  - robustness
  - scalability
- QAs inhibited
  - development costs

these are general considerations:
remember that a real analysis requires QA scenarios
on a concrete implementation strategy