

# Scaling and Anomalies in Massively Populated Persistent Worlds

Björn Knutsson

In cooperation with Honghui Lu (UPenn)  
and her students

# Massively Populated Persistent Worlds

- Large number of users: 1000 – 100000
- Applications include large-scale interactive simulations, training scenarios and entertainment



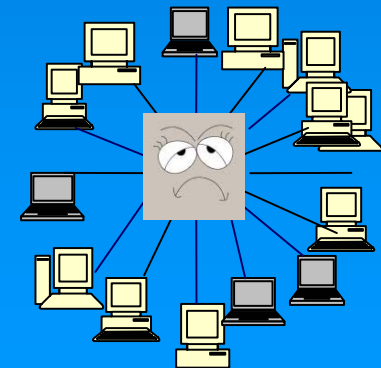
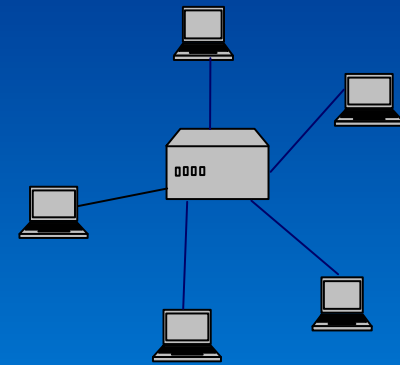
# DoD relevance

- DoD is currently buying this technology from the entertainment industry.
  - Army's PEO-STRI recently bought *There* engine to build *Assymmetric Warfare Environment*



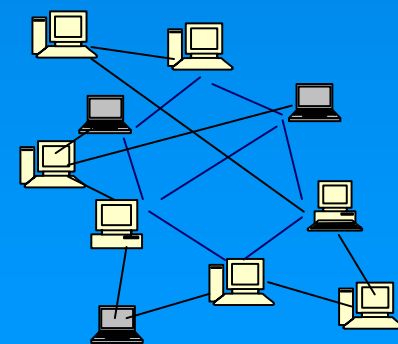
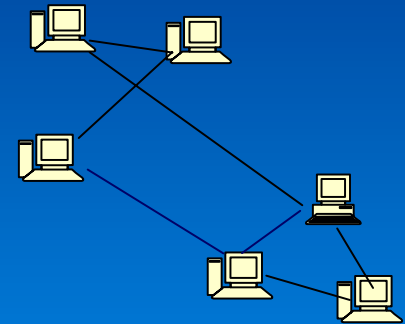
# MPPW architecture

- Designed for entertainment use:
  - Users pay subscription fee
  - Company needs full control over simulation
- Existing MPPWs are built using central servers or server clusters.
- Limits scalability, increase scale-up time and creates a single point of failure.
- Trust model: Only trust server



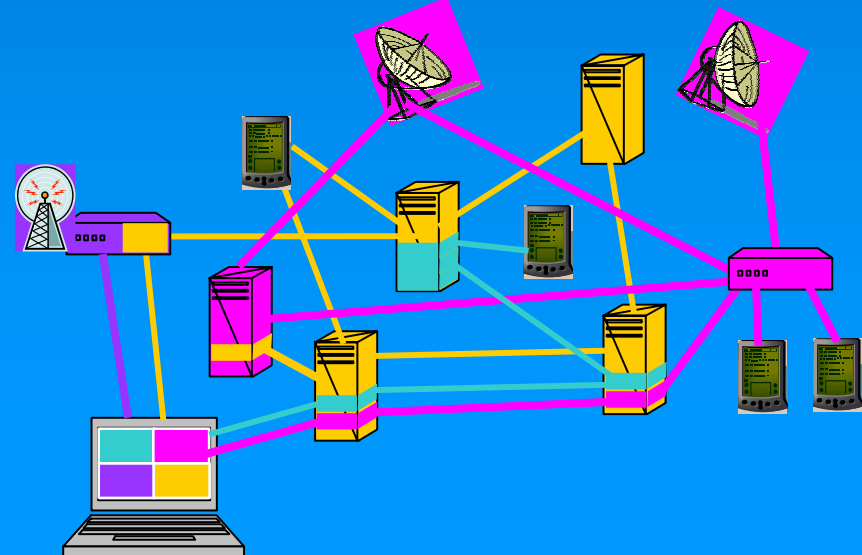
# MPPW architecture, cont.

- Missed opportunities: Clients possess significant resources that are not utilized
- Our goal: Exploit implicit diffuse computing infrastructure available
- Trust issue: Can clients be trusted?
  - Our first take: Assume client software can be trusted, but user operating client cannot.
  - Parallel track: Anomaly detection



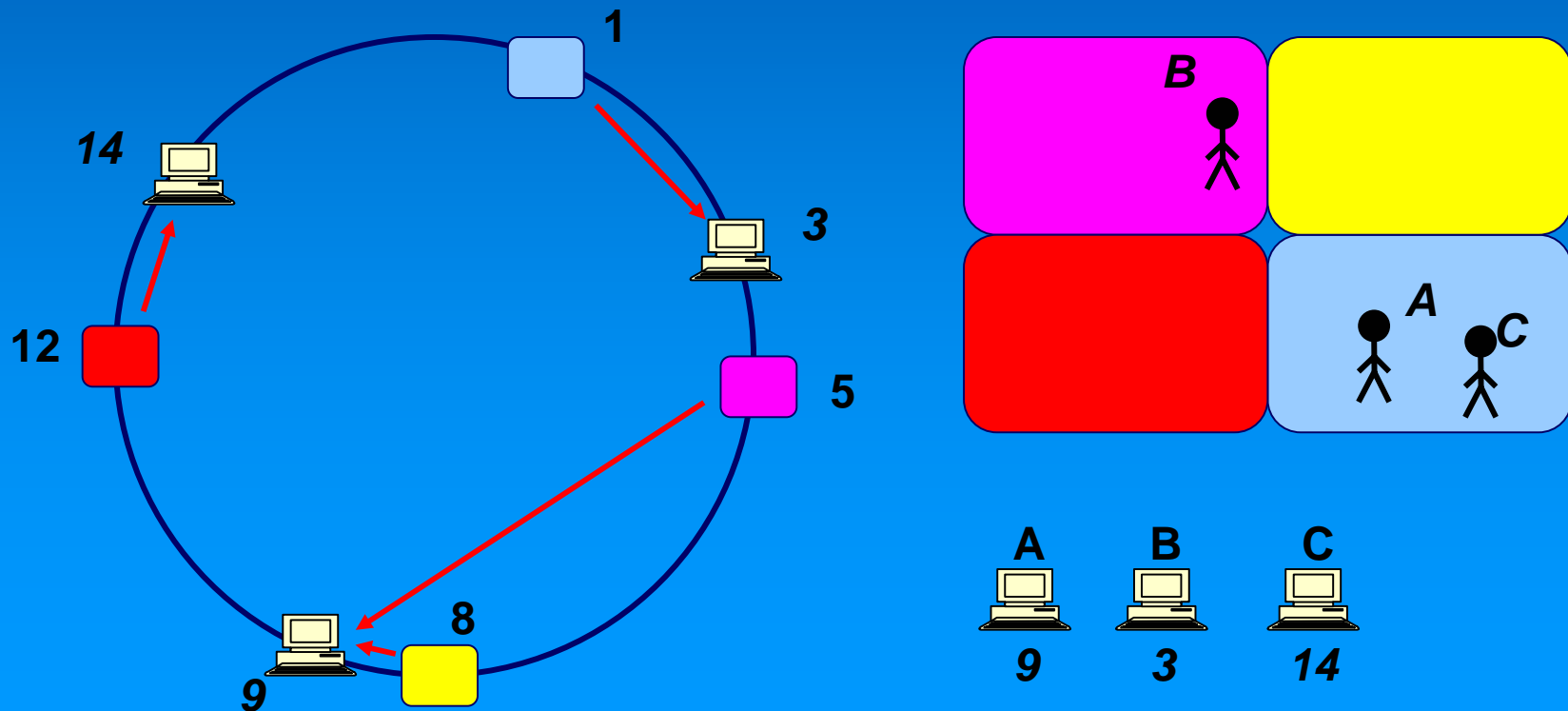
# Scaling using diffuse infrastructure

- The clients currently participating in the world can be used to also manage the world
- Available resources scale naturally with number of active participants consuming resources
- Must cope with both exits and disconnects
  - Automatic recovery
  - Automatic reconfiguration
  - Conserve bandwidth



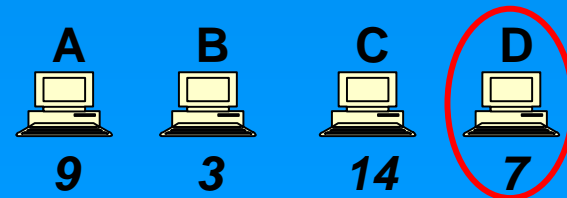
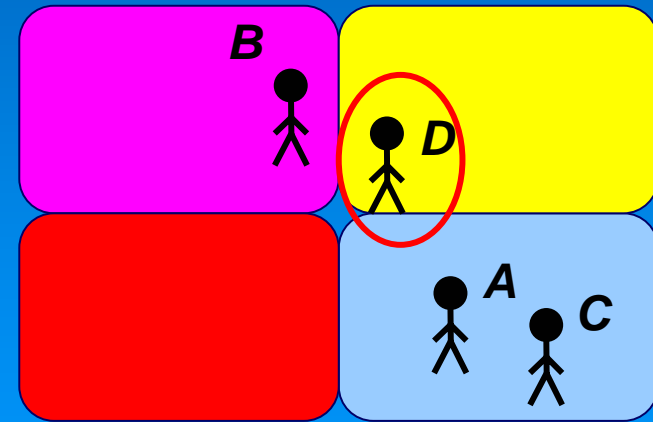
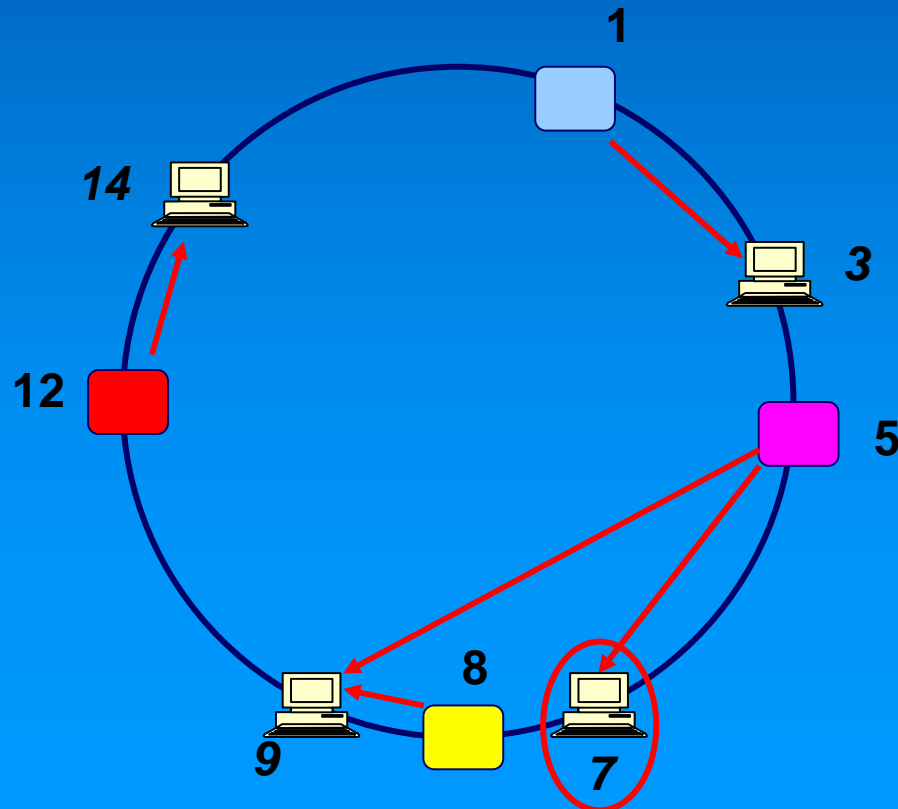
# State distribution

- Assign map regions and clients random numbers
- Regions are managed by “closest” client



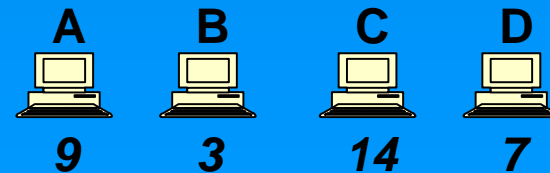
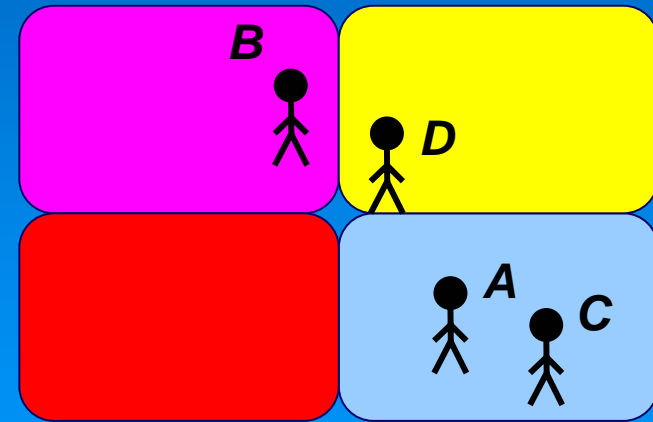
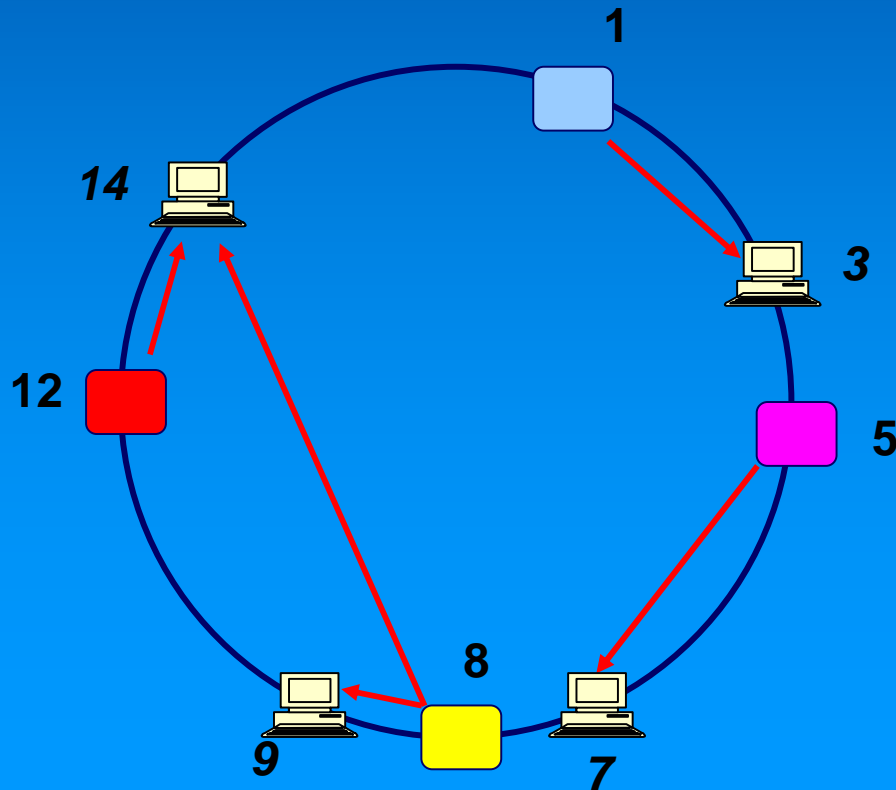
# Adding a participant

- Adding a participant also adds a node, which may cause a reconfiguration of responsibilities



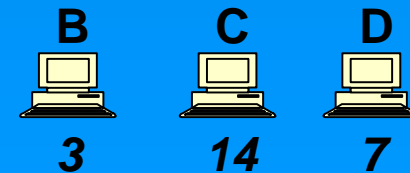
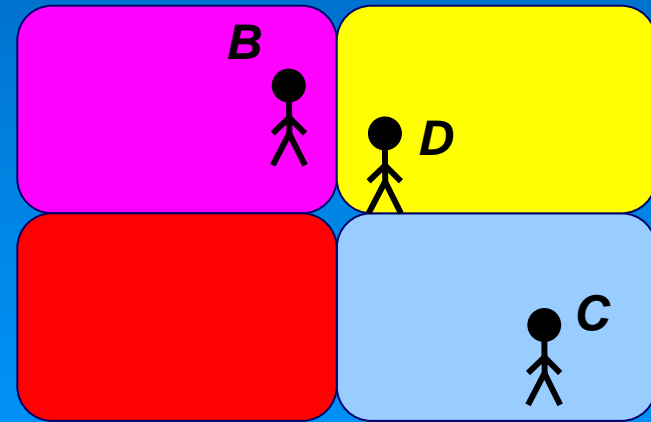
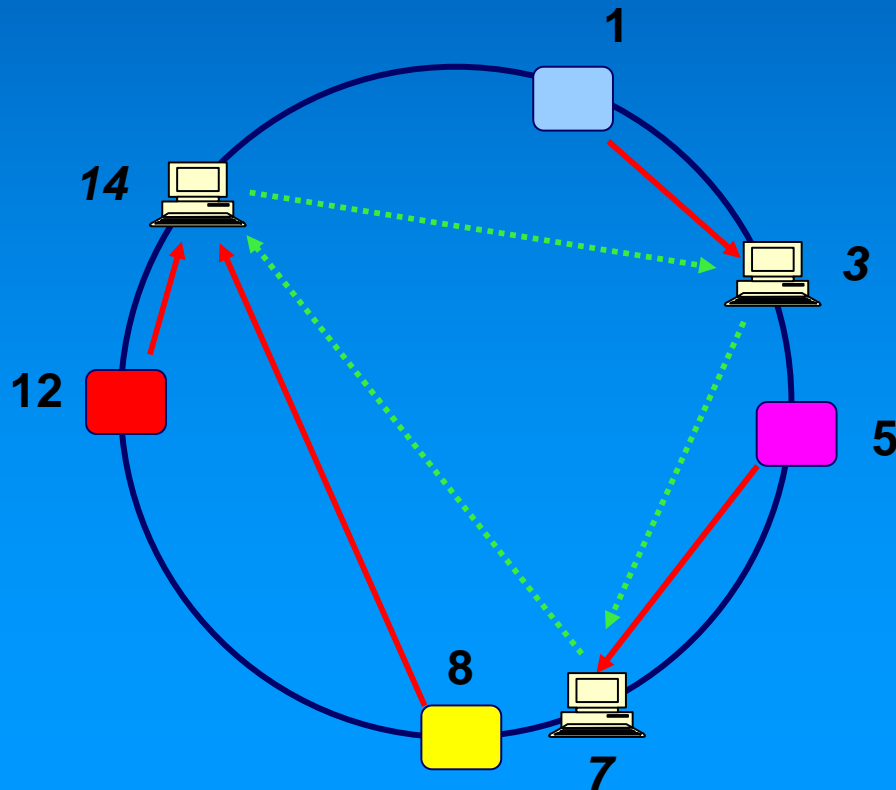
# Removing a participant

- The removal, or disconnection, of a participant can similarly cause a reconfiguration



# State replication and fail-over

- Data is replicated to avoid losses when nodes fail
- Exploit routing to make fail-over automatic



# Scalability

- Scalability is a function of density in regions, not number of participants or number of regions.
  - Interest management controls region size
  - Many participants = many nodes to use
  - Inactive regions are cheap to manage.
- Higher density allow more replication
- Peak density kept down by reducing region size
  - Balance between area of interest and region size



# Implementations

- Simulator prototype
  - FreePastry/Scribe
  - Simulates up to 4000 participating nodes
  - Described in Infocom'2004 paper
- New prototype – not simulator
  - Currently tested on Liniac cluster
  - Used for all our current subprojects
  - Designed to be independent of distribution mechanism

# Results from first prototype

- Stress-tested with average density of 10/25 participants per region and 1000/4000 participants
  - 6 updates per second, region change every 40 seconds
  - Interaction with objects/participants every 20 seconds
  - Per node load ~constant for 1000 and 4000 participants
  - Density major factor in per node load
  - Even frequent (1/second) node join/leave have little impact on overall performance
    - Even with single replica, state was lost very infrequently

# Anomalies in MPPWs

- Anomalies are unintended effects from rule set
  - Anomalies can be exploited to “cheat” or subvert
- Persistent state = persistent effects
- Participants cannot be assumed to be “honest”
  - In a training simulation, participants must be allowed exploit advantages – compare camouflage
  - In commercial uses, users may have monetary or other incentives to “cheat”, even if technically illegal

# Detecting (and preventing) anomalies

- Many sources, programming errors, malicious alterations etc
- We focus on server-side, high-level anomalies
  - Unintended interactions between different rules
  - Interactions between events and meta-events
    - Meta-world events are very interesting
  - High-level implementation flaws
  - Focus on detection, not prevention



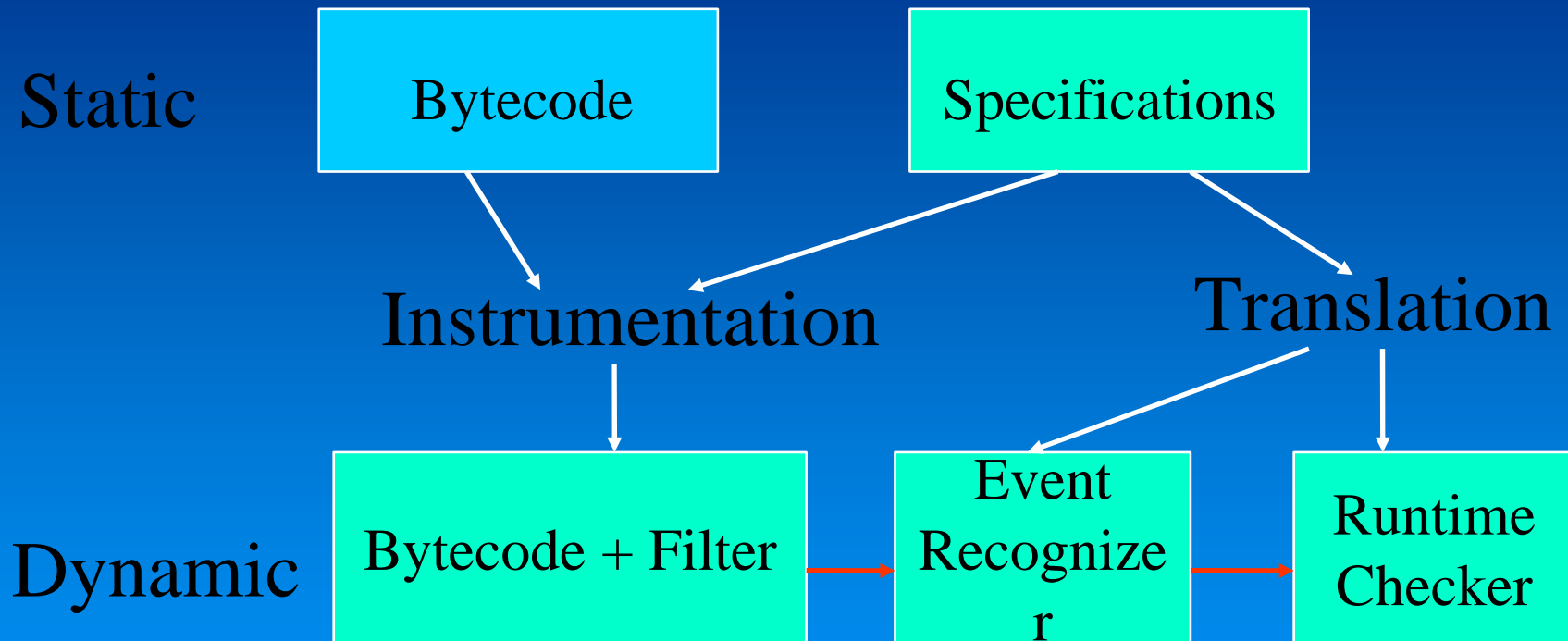
Meta-chess hamster event

# Run-time verification

- **Question:** Can run-time verification tools be used to detect high-level events in complex software?
- Instrument Java bytecode to find trigger events
  - Using Java-MaC tool, developed at UPenn
    - Cooperation with Insup Lee's group
- Detection in real-time and offline
  - Real-time results can “steer” program
  - Collect statistics, detect conditions
  - Does not prevent emergent behavior



# Java-MaC: Monitoring & Checking



- Open questions:

- Expressiveness and ease of producing specifications
- Performance and scalability to large systems

# Ongoing work

- Exploring different distribution approaches
  - Original approach – Peer-to-peer
    - Relies on #peers  $\gg$  #managed objects
    - Pseudo-random distribution – problems with heterogeneity
  - Virtual clustering
    - Cherry-picks resource-rich nodes into virtual cluster
    - Use more advanced load balancing techniques
      - Joint work with Cristiana Amza's group at University of Toronto
    - Exploring how “flocking” behavior affects load

# Ongoing work, cont.

- More work on run-time verification
  - Formalization of known problems
  - Use of machine learning
  - Overlap with intrusion detection?
- Seamless roaming between map regions
  - Dealing with overload
- Benchmarking commercial implementations
  - For comparison and simulations