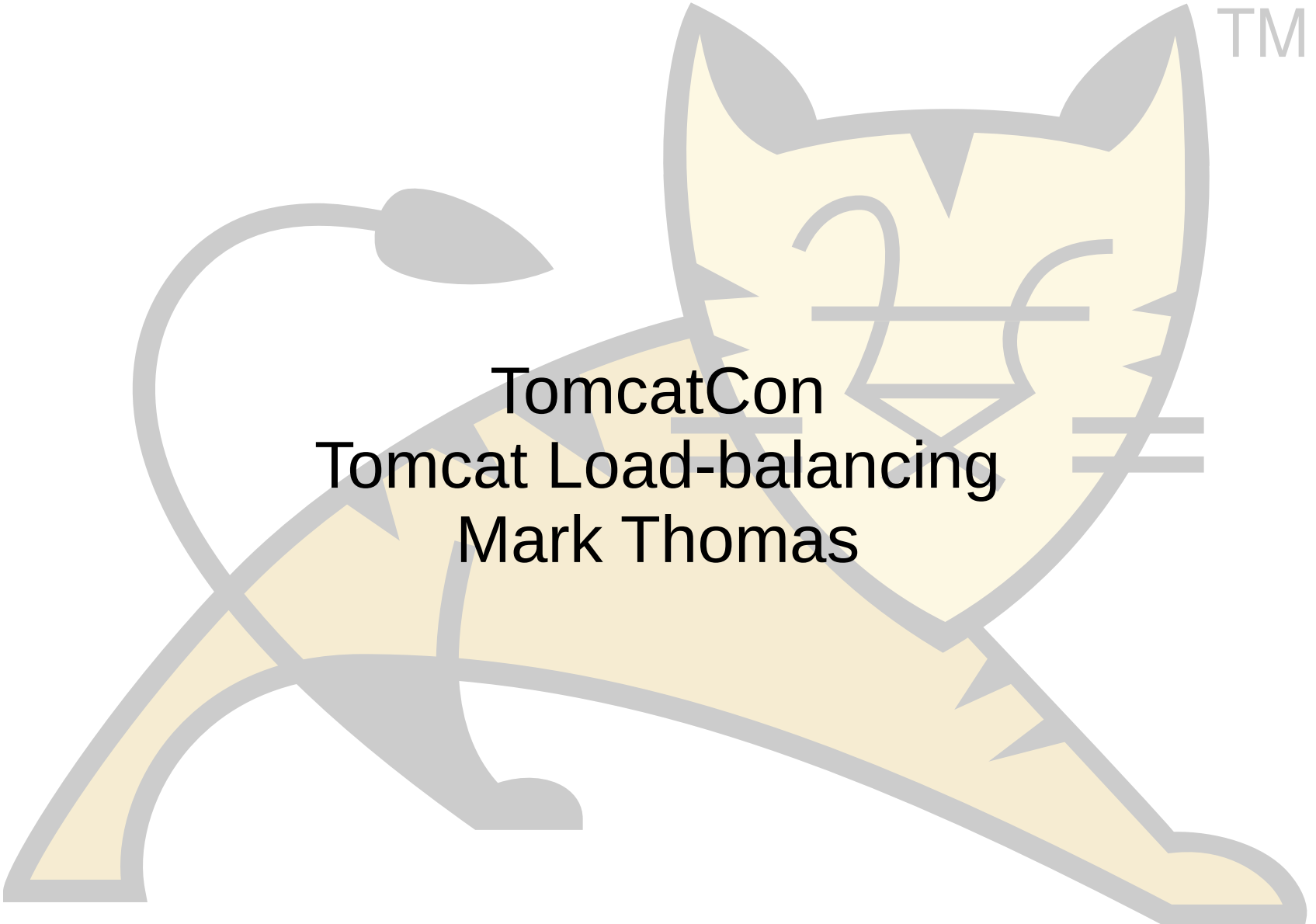
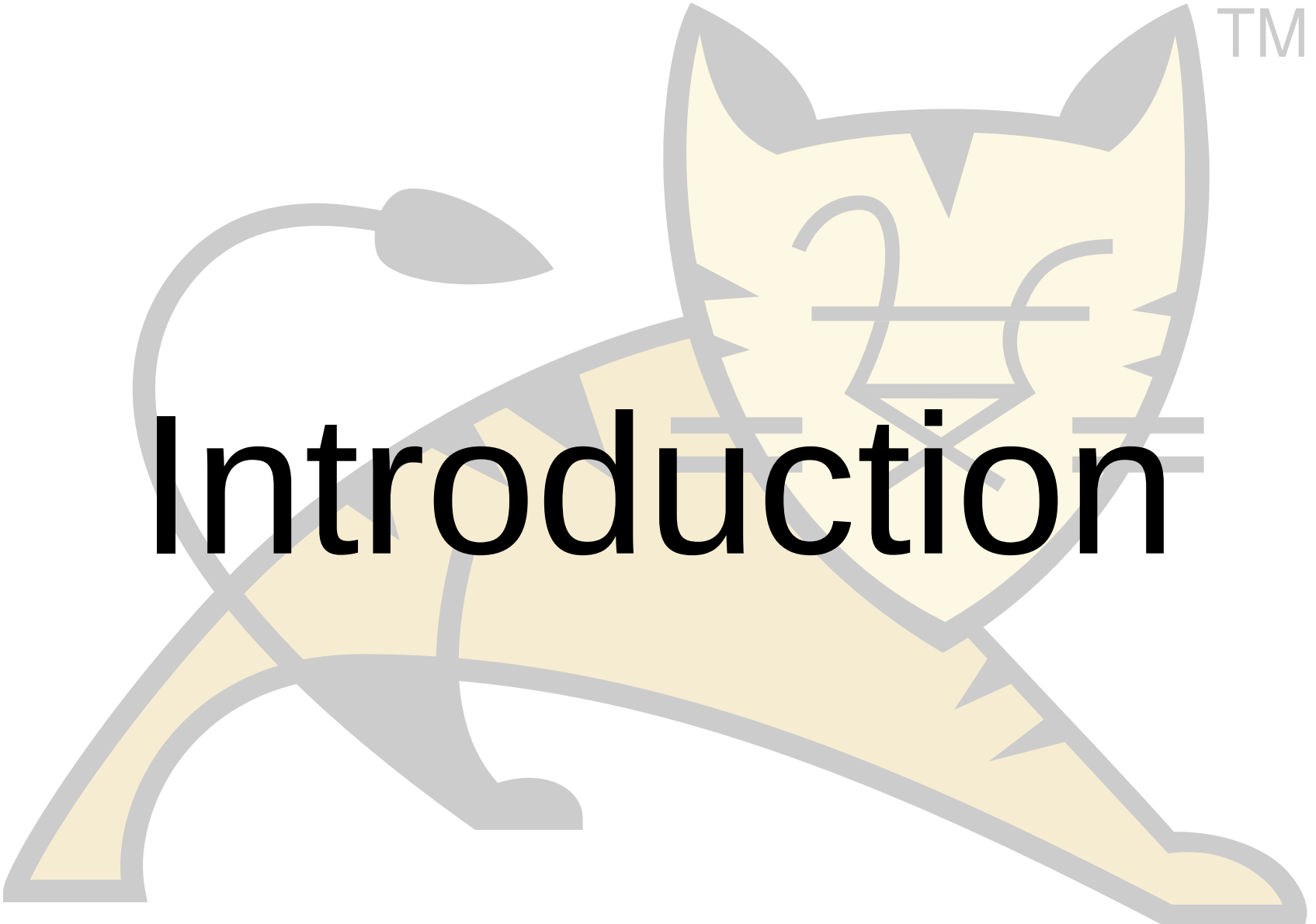
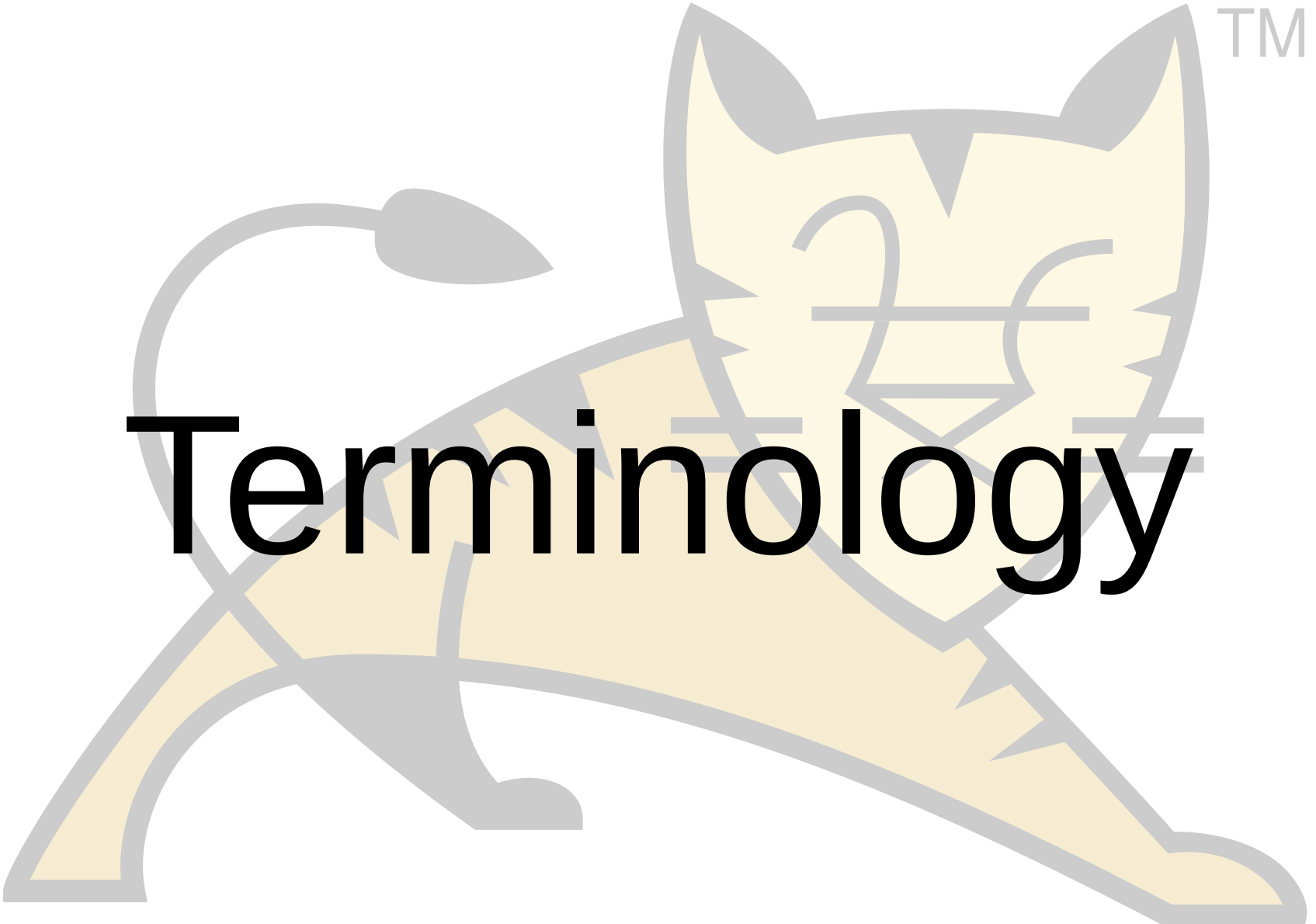


TomcatCon
Tomcat Load-balancing
Mark Thomas

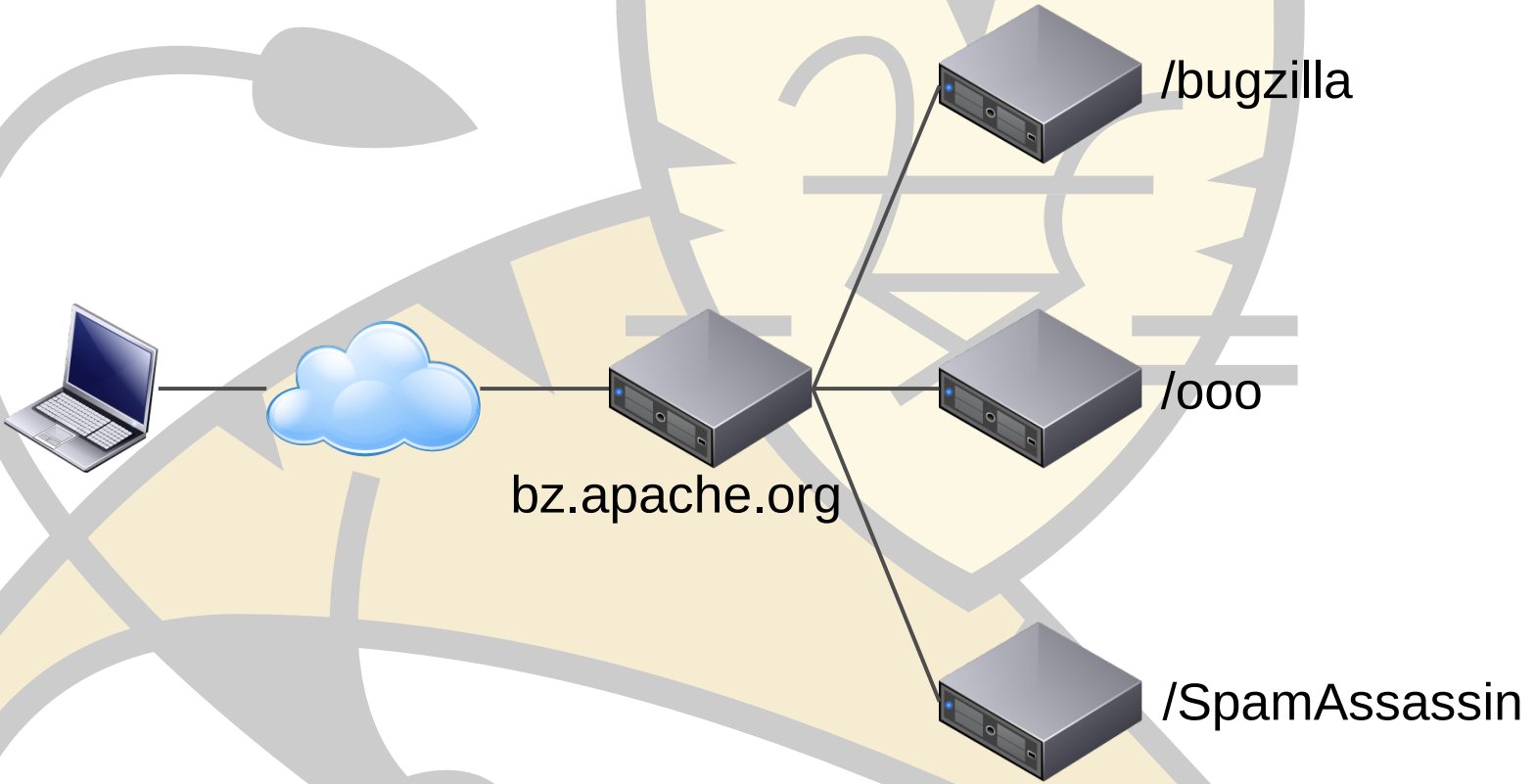


Introduction



Terminology

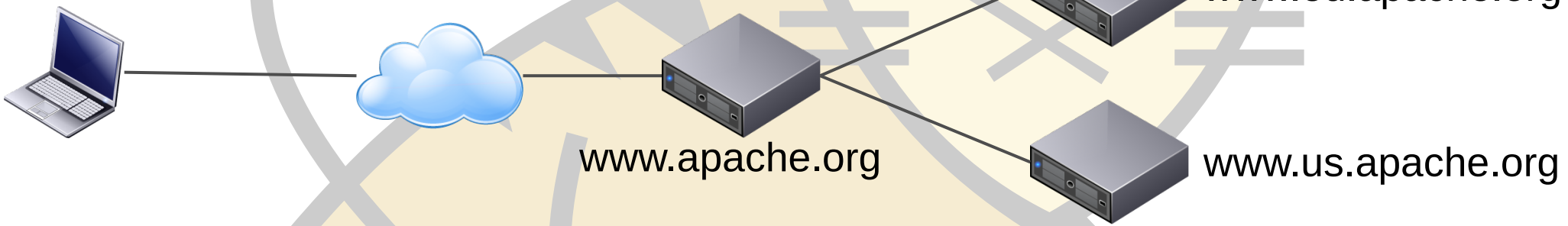
Terminology: Reverse Proxy



Terminology: Reverse Proxy

- Looks like a single host to the clients
- Usually multiple hosts
- Different services on different hosts
  - May also be geographically distributed
- Can be used to add features
  - e.g. Use httpd as a reverse proxy for Tomcat to add Windows authentication (no longer necessary)

Terminology: Load-balancing

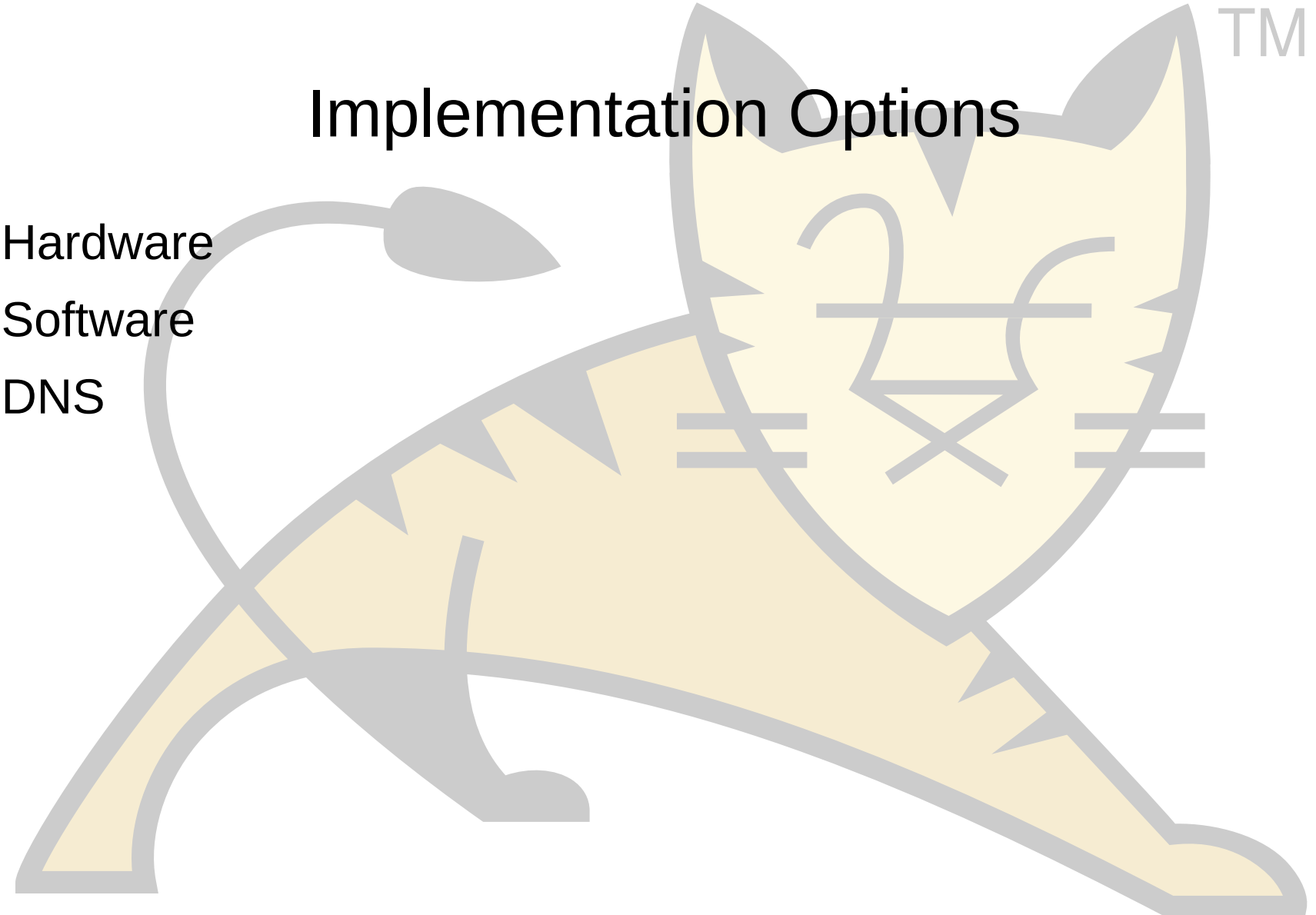


Terminology: Load-balancing

- Looks like a single host to clients
- Multiple hosts
- Each host is the same
- Each host is independent
  - No shared state between hosts
  - May share common services (authentication, database, etc.)
- Node failure may be visible to users

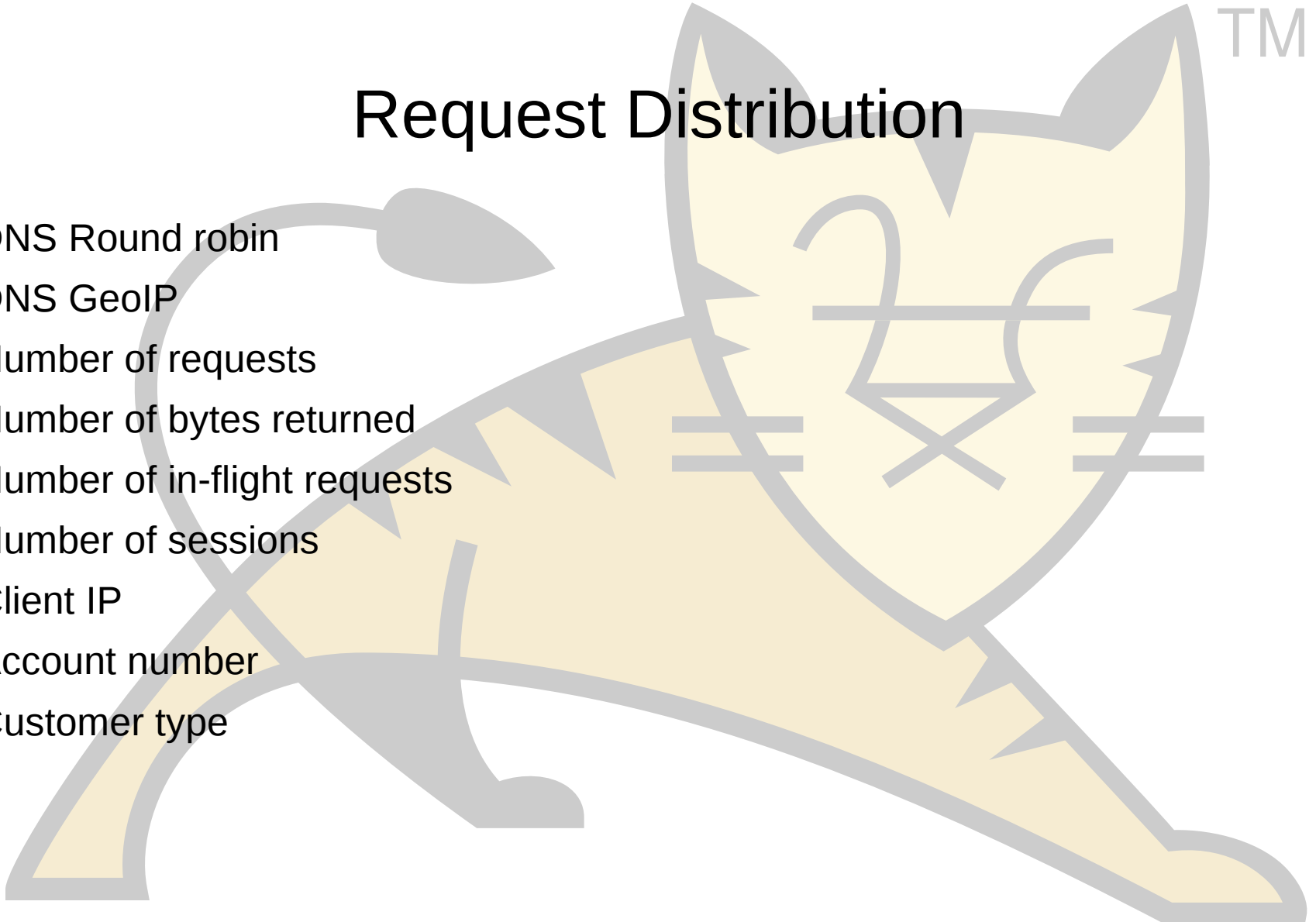
Implementation Options

- Hardware
- Software
- DNS



Request Distribution

- DNS Round robin
- DNS GeoIP
- Number of requests
- Number of bytes returned
- Number of in-flight requests
- Number of sessions
- Client IP
- Account number
- Customer type



Managing State

- Stateless applications are the simple solution
- Authentication implies state
  - HTTP session
  - Database
  - Request parameters
- Load-balancing is impacted by HTTP state

Terminology: Sticky sessions

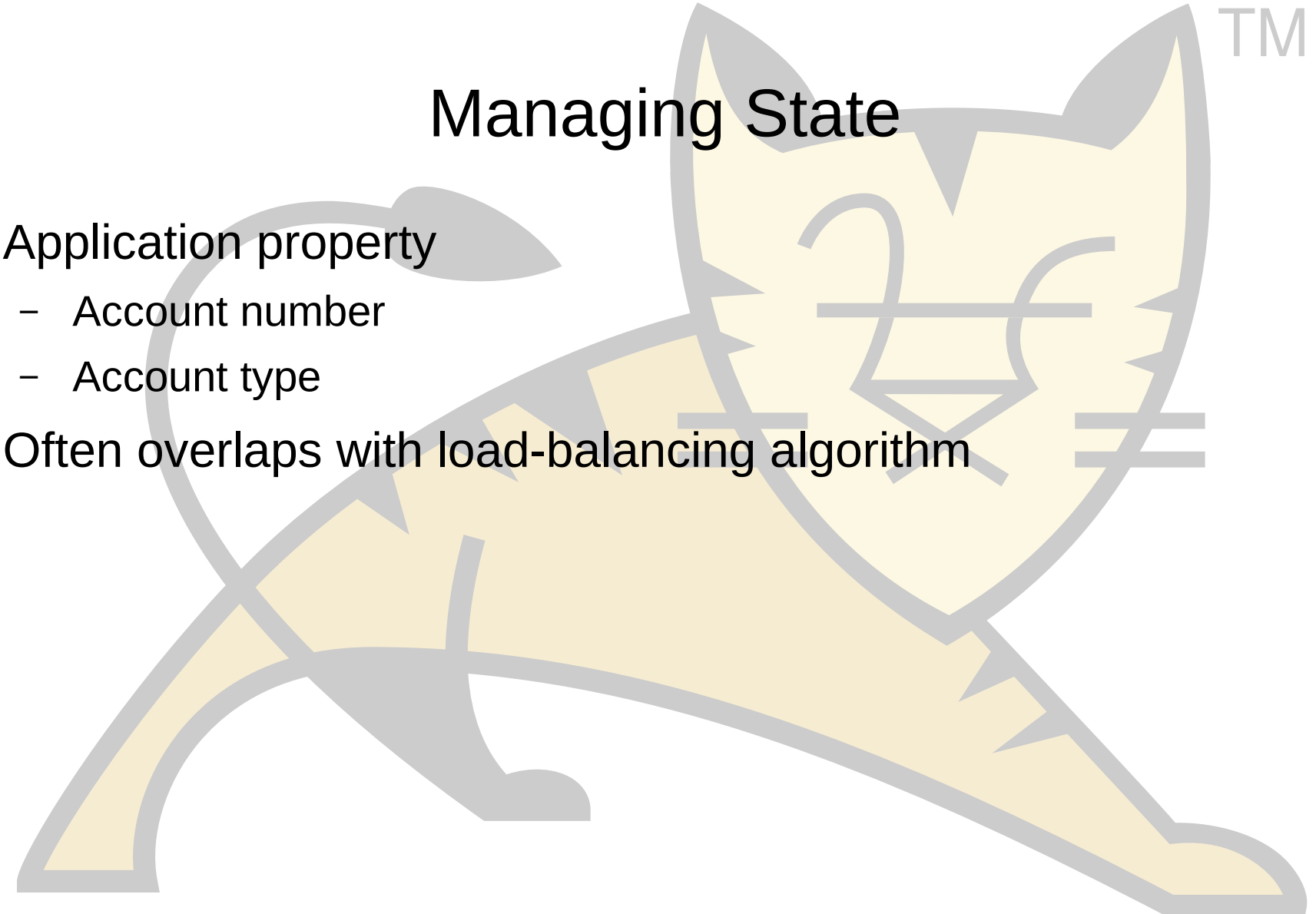
- Assume no clustering
- Session is created on node that handled request
- Load-balancer might send next request to a different node
  - Session won’t exist
  - Application will break
- Sticky sessions is a mechanism that ensures the subsequent requests are handled by the node the created the request

Managing State

- Sticky sessions are used for HTTP State
- Session ID
  - Something in the session ID identifies the correct node
  - Users could change this
- Dedicated cookie
  - Users could change this
- Property of client such as IP
  - Beware of ISP that use forward proxies

Managing State

- Application property
  - Account number
  - Account type
- Often overlaps with load-balancing algorithm



Failover

- Load-balancer needs to know the state of the nodes
- Nodes need to taken off-line for maintenance
  - Known in advance
  - Several options
- Nodes will fail
  - Not (usually) predictable
  - Need to be able to detect dynamically
- What is the impact on users?

Failover: Maintenance

- More transparent to users means
  - More complex configuration
  - Process takes longer
- Need to drain node of users
  - How long can an HTTP session last?
  - At what point do you stop the node anyway?
- Can Tomcat’s parallel deployment feature help?

Failover: Unexpected

- Typically there is no separate management channel between Tomcat instances and load-balancer
  - There is with mod_cluster from JBoss
- Need to detect failed nodes so failover can happen as early as possible

Failover: Unexpected

- Can use a ‘failed’ request to detect a failed node
- Is a 500 response because the server crashed or because of an application bug?
- Is a timeout because the server crashed or because it is just a long running request?
- Applications that can have long running requests take at least that long to detect failures.

Failover: Unexpected

- Monitoring user initiated requests to detect node failure is fragile
- Load-balancer triggered request to known, working, ‘simple’ page
  - More reliable
  - Still an HTTP request with the associated overhead
- Protocol pings are even faster

Questions