State spill is the underlying cause

State spill is the act of a software entity’s state undergoing a lasting change as a result of handling a transaction from another entity.

Classification of state spill

Based on four common OS entity design patterns:

- **Indirection Layers** convert between high-level and low-level representations of data and commands.
  - Virtual File System abstraction
  - Process abstraction
  - Microkernel userspace servers
  - Device drivers

- **Multiplexers** temporally or spatially share an underlying resource among multiple clients.
  - Schedulers / process mgmt
  - Window managers
  - High-level drivers

- **Dispatchers** register client callbacks to properly deliver events or messages.
  - Device event callbacks
  - Synchronization primitives
  - Upcalls
  - IPC layers

- **Inter-Entity Collaboration** requires synchronization of non-orthogonal states to ensure correctness.
  - Microkernel userspace servers
  - Android services

Designs to avoid state spill

- Client-provided resources
- Stateless communication
- Hardening of entity state
- Modularity without interdependence
- Separation of multiplexing from indirection

State spill in Android system services

- **STATESPY** found state spill in 94% of Android services analyzed, most with 1-10 instances
- Classified state spill instances in 60 transactions:
  - 39% caused by indirection layers
  - 21% caused by multiplexers
  - 55% from dispatchers/collaboration
- Better discovery of problems in app migration than manual identification of residual dependencies
- Discovered secondary spill in 27 services:

Entity granularity dictates state spill

State spill is relative to the chosen entity granularity. Low-level entity interactions (shaded) are unimportant.