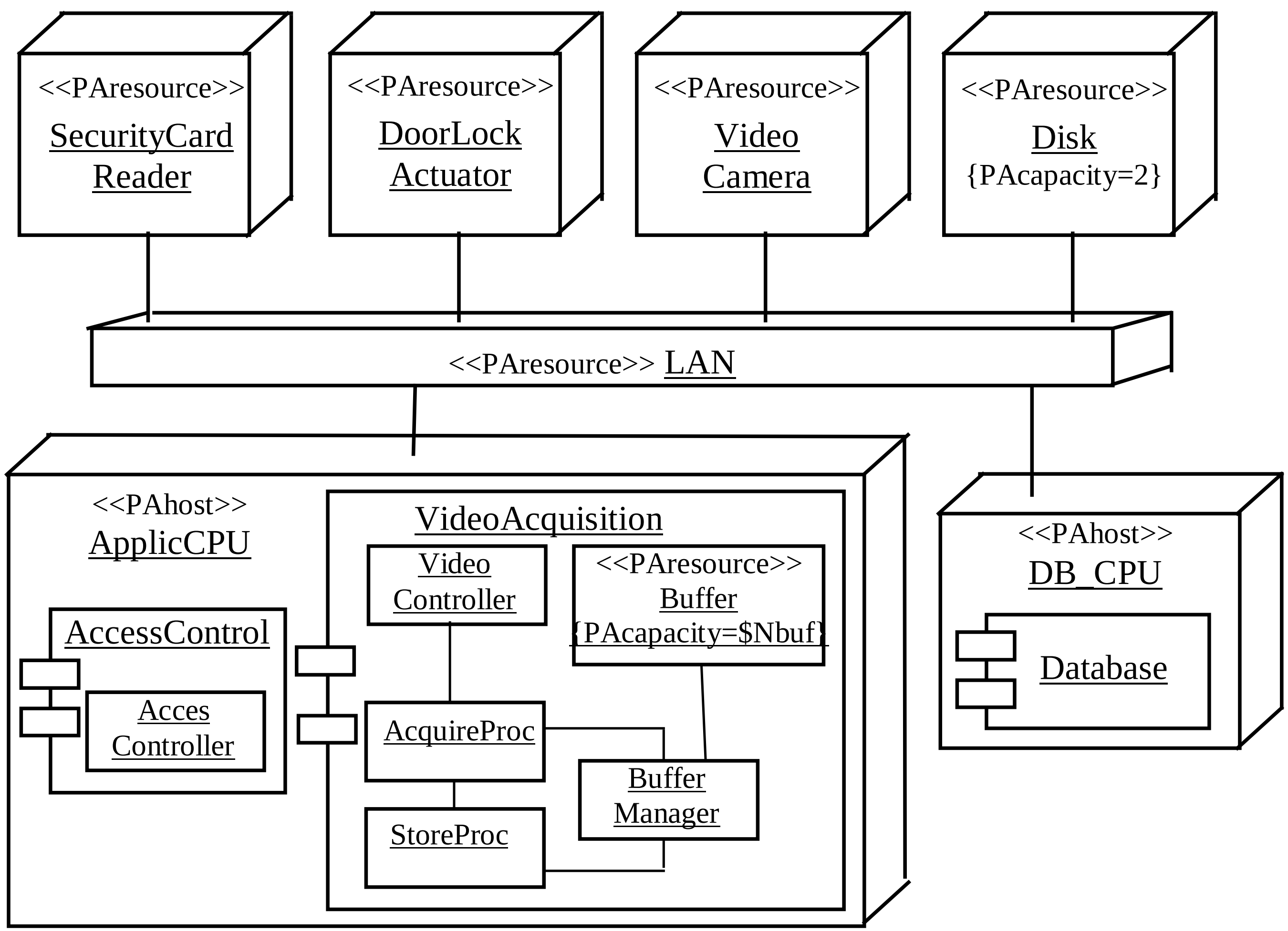
Layered Queueing Network Modeling of Software Systems

Building Security System (buffering)

Murray Woodside
5201 Canal Building

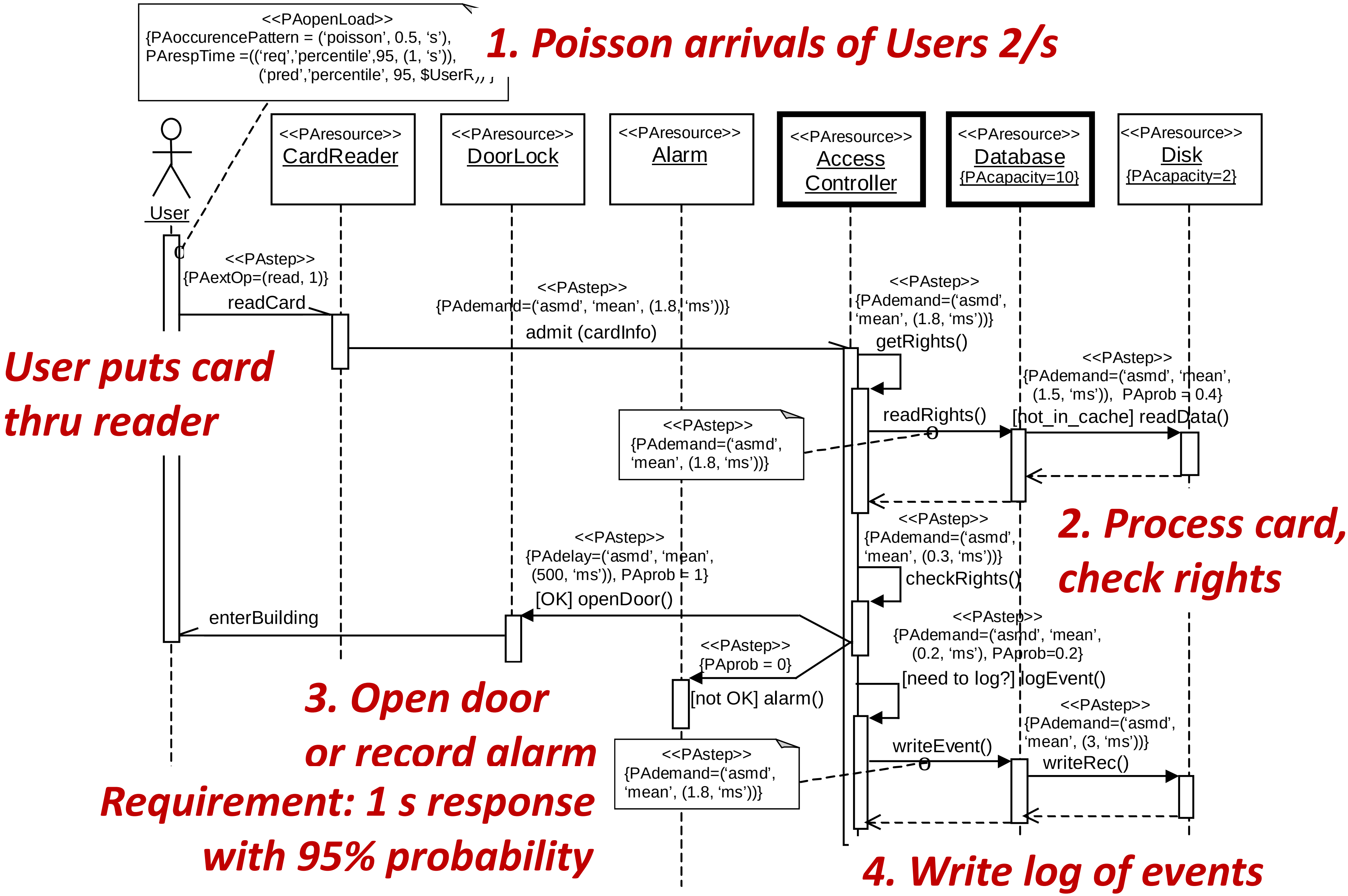
Building Security System

- Two subsystems: CCTV storage, and door access control
- hope to manage up to 100 cameras
- Components, shown as UML with MARTE annotations:



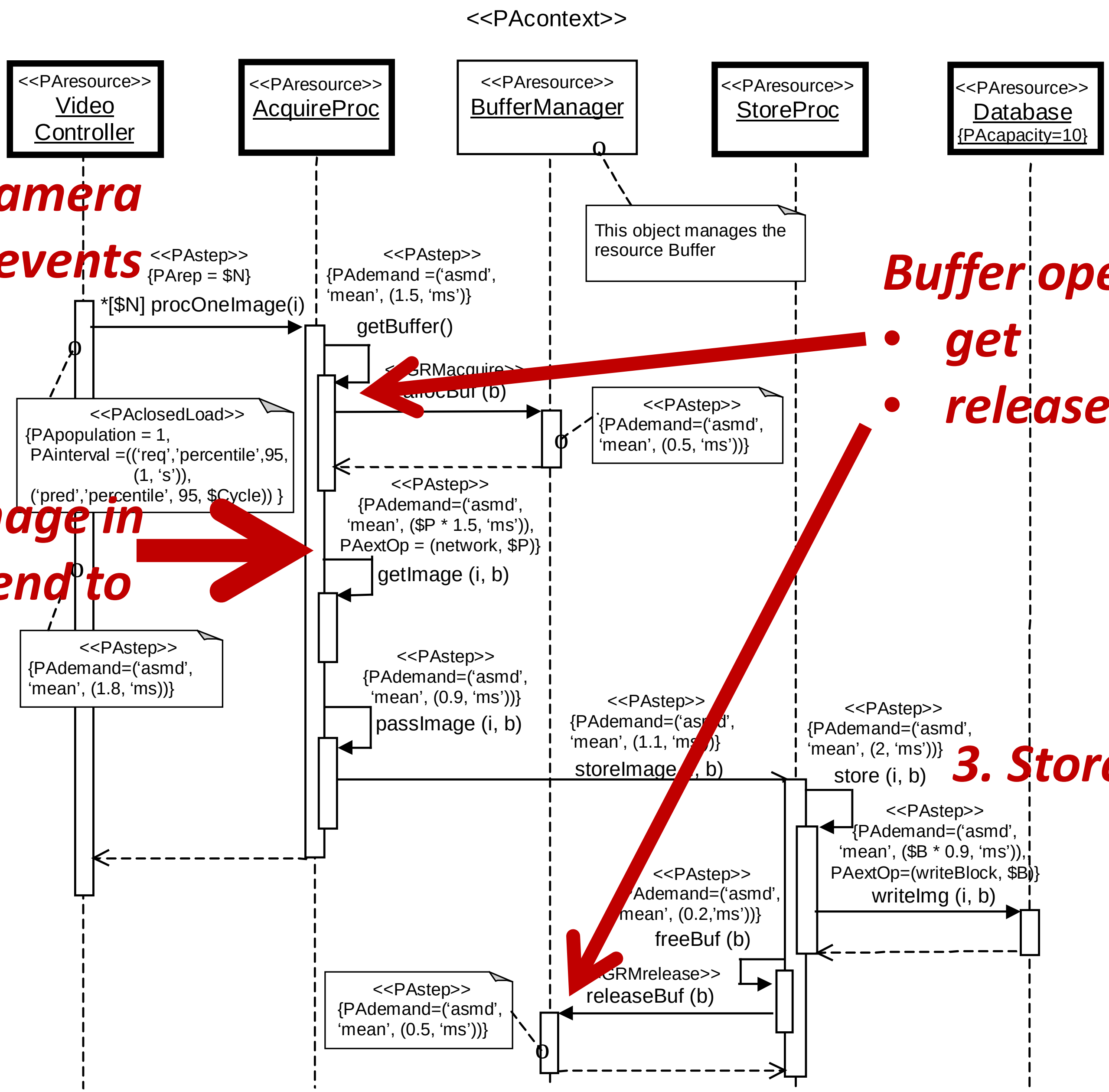
Carleton University
Canada's Capital University

Door Access Scenario



Carleton UNIVERSITY
Canada's Capital University

CCTV capture scenario



1. Trigger camera read events

(the N cameras were not modeled)

2. Put the image in a buffer, send to database

Buffer operations:

- ***get***
- ***release***

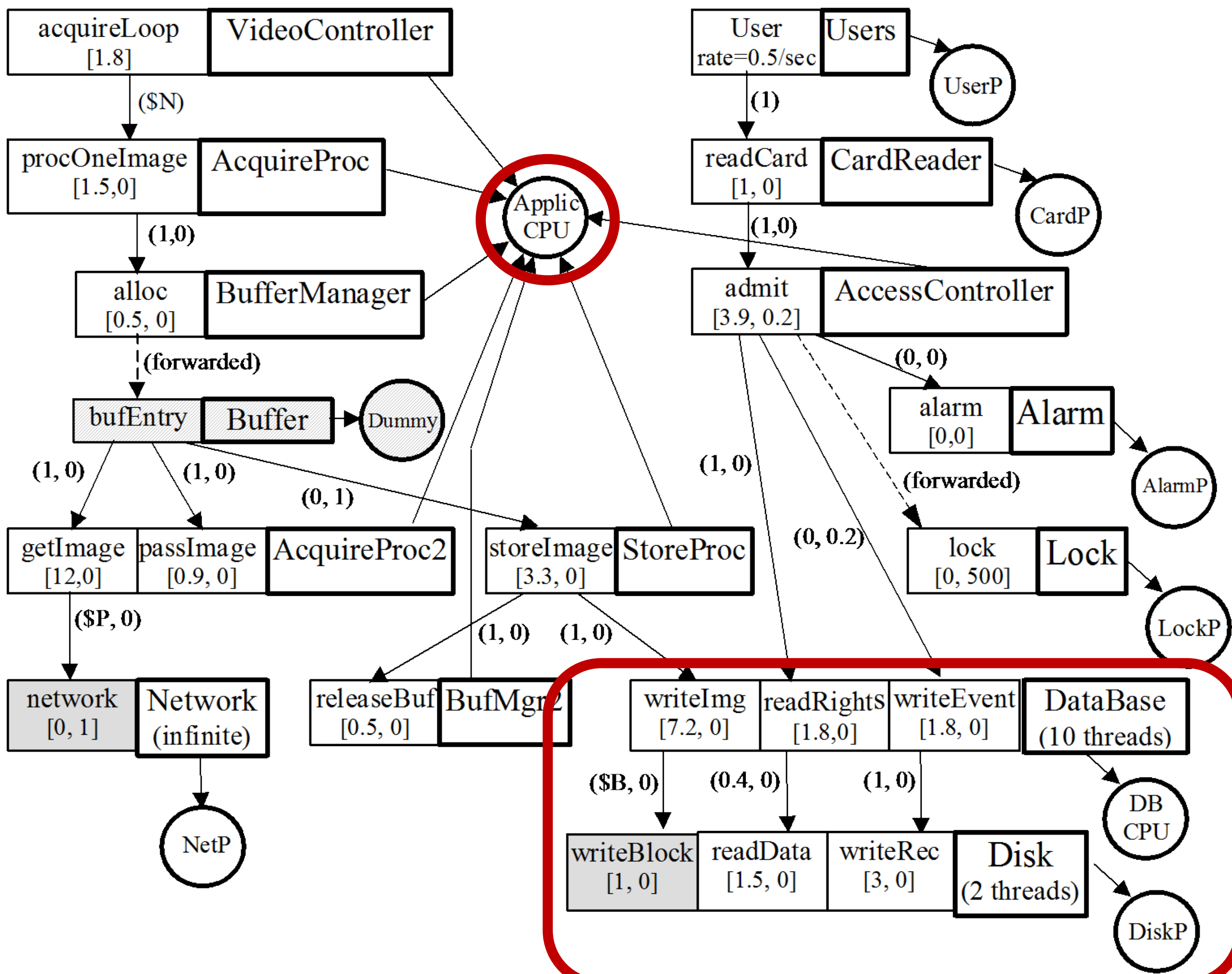
3. Store image in database

The LQN Model

Shared Resources

Video Capture subsystem

Door Access subsystem

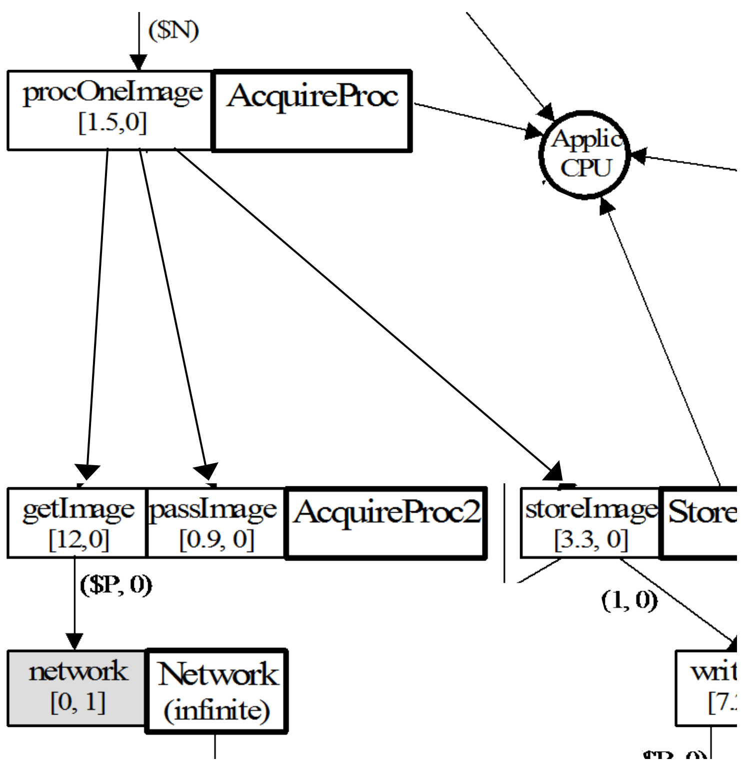


Handling of Buffering

Each buffer must be emptied before it can be used for another camera

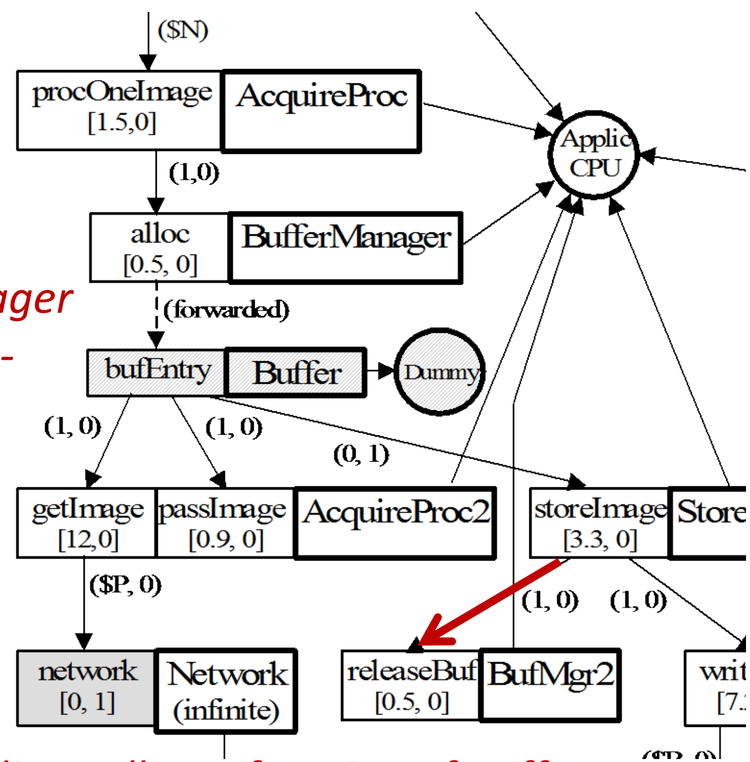
- Thus the buffer is a resource that could have a queue, which should be modeled as the pseudo-task Buffer

Model fragment without buffer



How the buffer pool was modeled

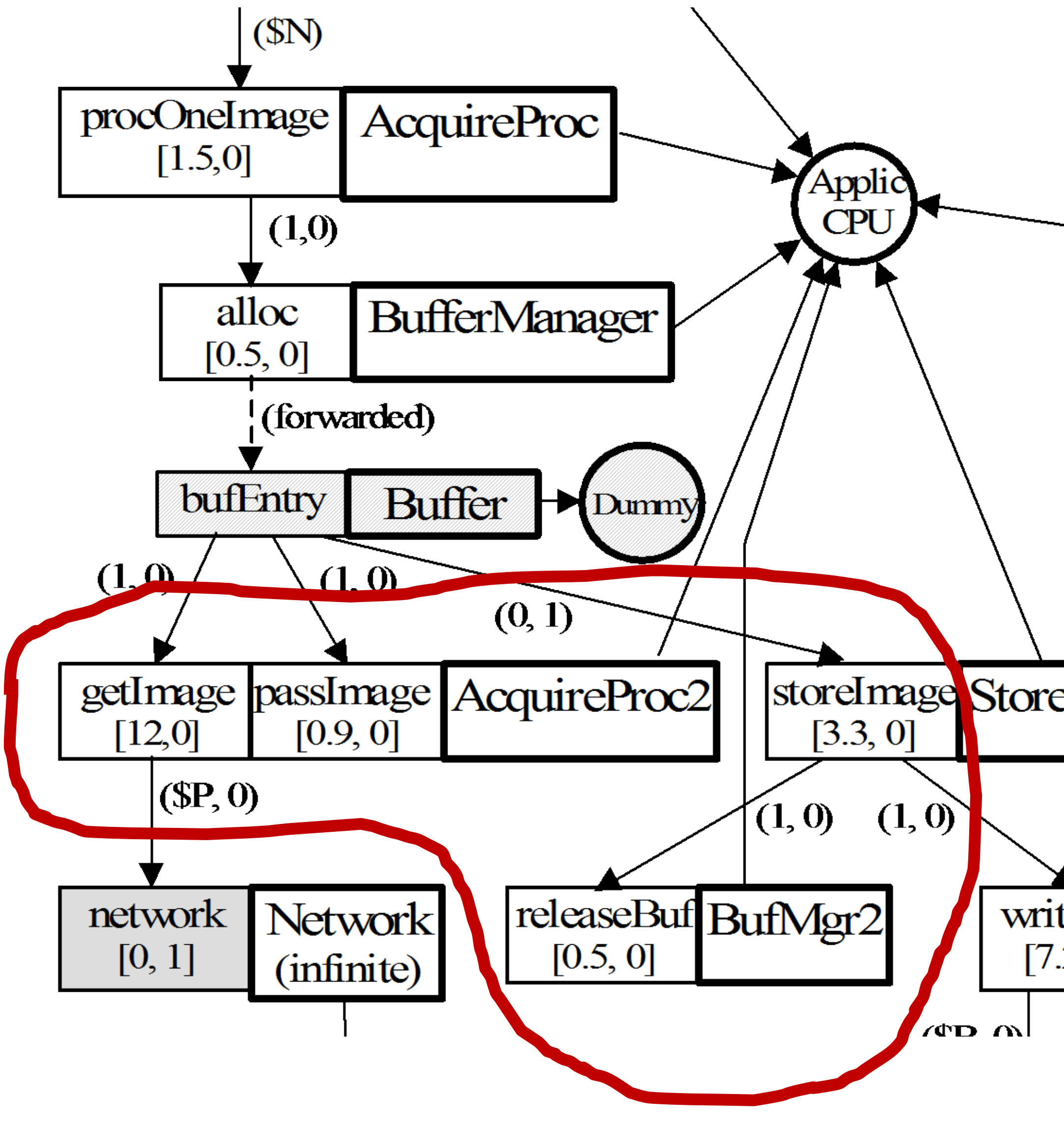
real task BufferManager and pseudo-task Buffer



forwarding call to a function of BufferManager

Handling of Buffering (2)

- The operations that require holding the buffer are executed by calls from the buffer pool pseudo-task **Buffer**
  - –separate from the buffer manager task!!
  - –including the execution of the release operation by the buffer manager
  - –assumes the manager has a dedicated thread for release
- releaseBuf is executed by storeImage, or bufEntry



Model Features

This model illustrates

- how we can model logical resources (the buffer pool)
- the use of forwarding
- the use of second phase to improve concurrency (later)

Results #1

Ncam	Average Response Time		Normalized Utilizations				Prob of Missing Deadline	
	Cycle (sec)	User (sec)	AcqProc	Buffer	StoreProc	AppCPU	Cameras	Doors
10	0.327	0.127	0.960	0.9998	0.582	0.549	0	0.031
20	0.655	0.138	0.963	0.9999	0.582	0.545	0.0007	0.036
30	0.983	0.133	0.964	0.9999	0.582	0.544	0.4196	0.038
40	1.310	0.129	0.965	0.9999	0.582	0.544	0.9962	0.034

Table 1. Simulation results for the base case

- Base case, one buffer, so one camera at a time
- Access-control responses are fine; the event rate was kept constant at 2/s.
- Camera polling becomes too slow between 20 and 30 cameras
  - try adding more buffers.

Results #2

- cameras fixed at 40, vary the number of buffers NBuf
  – disappointing: the miss probability levels out above 9% at about 7 buffers.
- StoreProc is apparently the new bottleneck: try an additional thread

NBuf	Average Response Time		Normalized Utilizations				Prob of Missing Deadline	
	Cycle (sec)	User (sec)	AcqProc	Buffer	StoreProc	AppCPU	Cam's	Doors
1	1.309	0.137	0.965	0.9999	0.583	0.544	0.9961	0.034
2	1.016	0.132	0.975	0.8762	0.800	0.702	0.5503	0.032
3	0.941	0.132	0.980	0.8235	0.893	0.756	0.2506	0.036
4	0.911	0.131	0.983	0.8042	0.936	0.782	0.1597	0.032
7	0.879	0.132	0.986	0.8136	0.984	0.810	0.0948	0.033
10	0.872	0.129	0.987	0.8437	0.995	0.817	0.0935	0.034

Results #3

Success!

- Two threads on StoreProc combined with 4 buffers brings the miss probability down well within spec of 1 second for each

40 cameras, Nuser = 100 doors, Nbuf = 4 buffers

No. of Store Proc	Average Response Time		Normalized Utilizations				Prob of Missing Deadline	
	Cycle (sec)	User (sec)	AcqProc	Buffer	StoreProc	AppCPU	Cam's	Doors
1	0.911	0.131	0.983	0.8042	0.936	0.782	0.1597	0.032
2	0.756	0.137	0.946	0.5805	0.616	0.940	0.0022	0.035
3	0.743	0.139	0.932	0.5484	0.441	0.956	0.0015	0.039

Discussion

- The saturated resource is AppProc
  - making multiplicity = 2 allows 50 cameras
- The limitation is now at AcquireProc, due to a long service time
  - the time it takes to store the buffers is limiting
  - multithreading alone is not the answer
- To allow an earlier start on the next camera, we can put the calls from AcquireProc into phase 2, with multithreading
  - early reply to VideoController moves the capture on to the next camera much earlier
  - allows the concurrent phase-2 Acquire tasks to run in parallel
- Other adjustments are also possible

Results #4

- By using phase 2 at AcquireProc and various multiplicities we can get a capacity of 100 cameras.
  - Even more capacity can be found with StoreProc.
- Another exploration approach: set multiplicities at inf and see if specified delays are feasible at all, and what mulitplicity is used (= utilizattion), then work down to specified delays.

100 cameras, Nuser = 100 doors, Nbuf = 10 buffers

Multiplicity (Acquire, Buffer, Store, App. CPU)	Average Response Time		Normalized Utilizations				Prob of Missing Deadline	
	Cycle (sec)	User (ms)	Acquire Proc	Buffer	Store Proc	App CPU	Cam's	Doors
2, 4, 2, 2	1.250	0.133	0.988	0.923	0.886	0.710	0.9995	0.0332
2, 10, 6, 3	0.837	0.132	0.988	0.689	0.751	0.707	0.0057	0.0307
3, 10, 6, 3	0.768	0.134	0.983	0.895	0.910	0.769	0.0005	0.0352