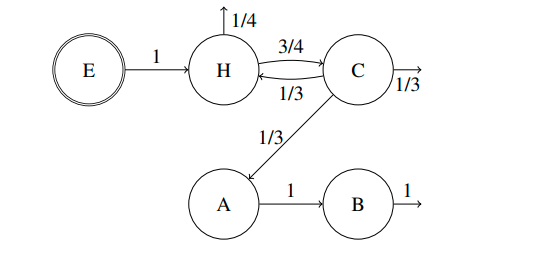
1)

ai)

UBG Tutorial Question3



ii)

VH = 4/3

VC = 1

VA = ⅓

VB = ⅓

iii) L = sum of visit ratios = 3

iv) 1.25.

U = lambda \* (0.1 + Vh + 0.2 \* Va + 0.3 \* Vb + 0.5 \* Vc).

U = lambda \* 4/5

Set U = 1 (max utilisation of 100%) and solve for lambda.

b) Pseudocode:

1. Initialize centroid positions randomly
2. Repeat until convergence of the centroid positions
   1. For every point, assign it to the cluster with nearest centroid
   2. For every cluster recalculate the centroid position

K-means clustering helps by clustering users into groups, each modelled by a centroid. Since each centroid is a point in the Euclidean space that maps to a UBG, the UBG associated with the centroid can be used for load testing.

2) a)(i) y = q\_0 + q\_ax\_a + q\_bx\_b + q\_abx\_ab

The effects are coefficients that explain the influence of the factors on the response y and the interactions among them.

(ii) q\_0 = 9, q\_A = -1, q\_B = -0.5, q\_AB = 0.5

so, SSA = 4, SSB = 1, SSAB = 1 and SST = 6

so, % var due to A = 66.7%

% var due to B = 16.7%

% var due to AB = 16.7%

(iii) lolz wtf

generator: D=BC

B=BC -> BD = C

D = BC -> BC = B

are there more confoundings? and what's the name of the design? idk pls help

2(4-1) design

Generator: D = BC

Other confoundings:

I = D\*D = BCD

A = ABCD

B = CD

C = BD

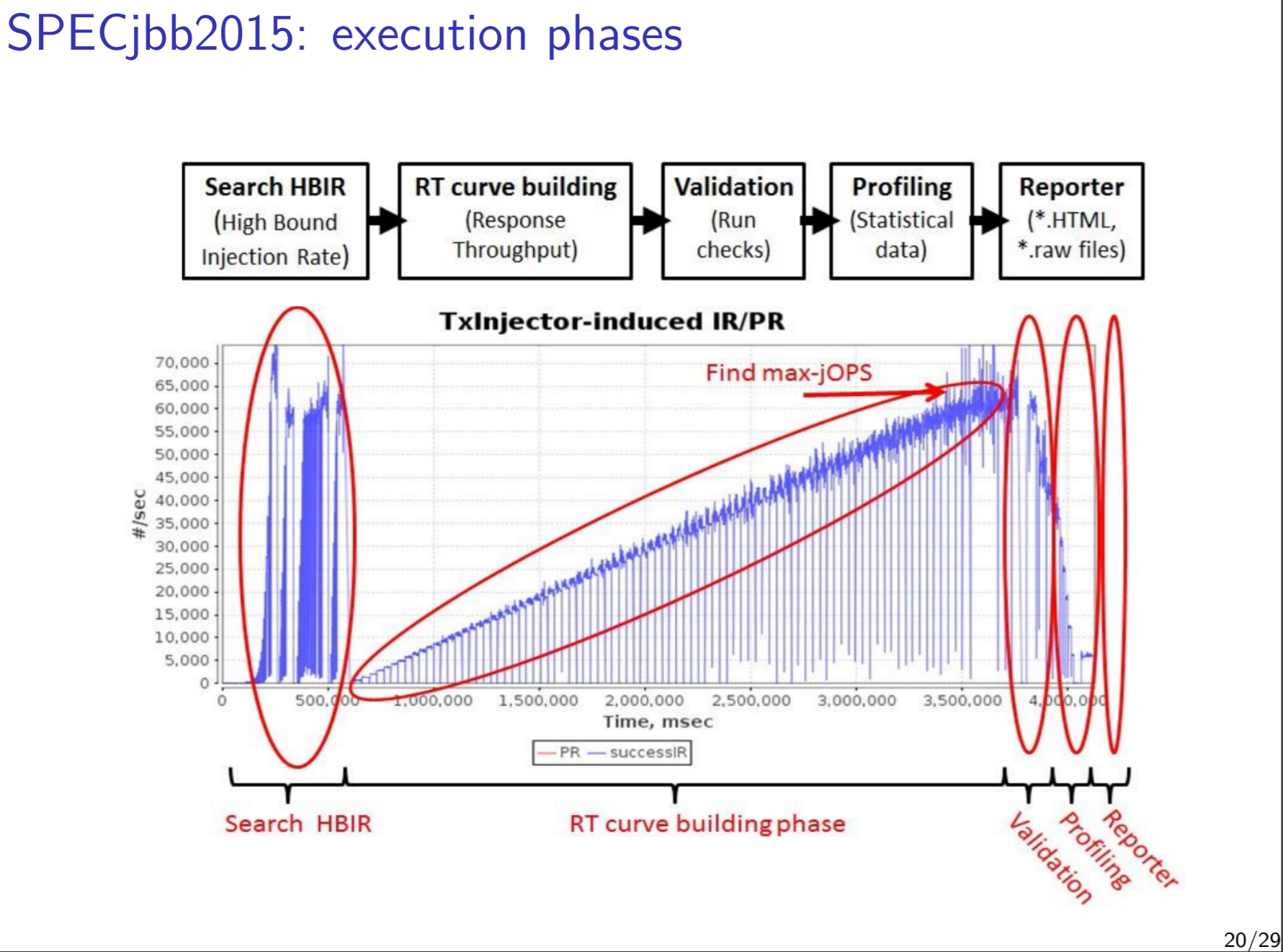
AB = ACD

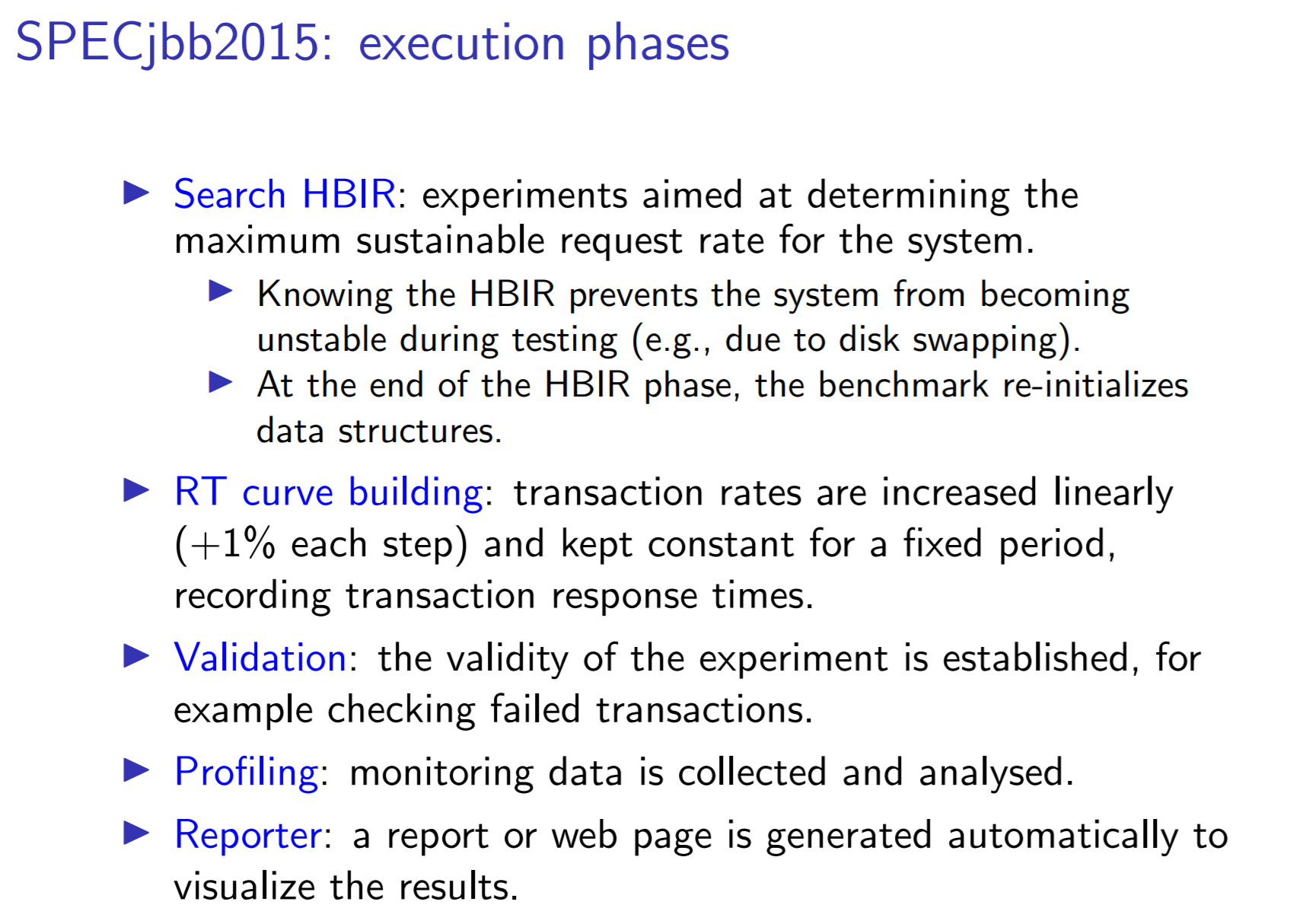
AC = ABD

ABC = AD

(iv) This is a resolution III design. 3/10 would not recommend

2) b)



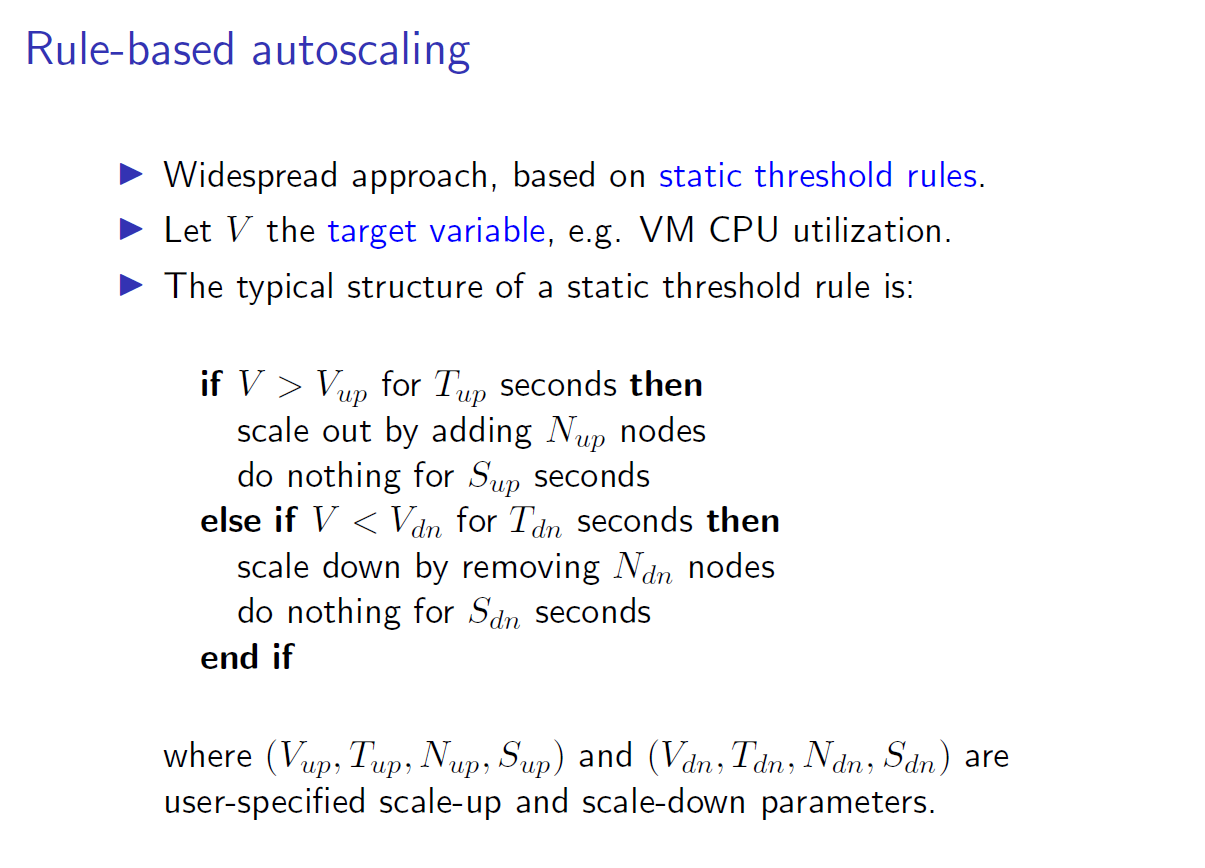


3)

Ai) A cloud native application relies on stateless autonomous compute nodes. This makes scaling up to 200 nodes as easy as scaling up to 2. As such, these resources can be automatically be provisioned on a needs-be basis for when demand for the service scales quickly. This eliminates the time needed to spend on provisioning these nodes.

Aii) Design the backend as a set of homogenous nodes (e.g microservices) that act as workers to new incoming requests for the service that they are looking after. This will then allow for the backend application to perform horizontal scaling.

Aiii)



Bi) True – a resource can be a potential bottleneck, depending on the mix of active requests. As such, linear programming needs to be done to find when this happens as we vary the payload of requests.

Bii) False – CPU main memory can be the bottleneck e.g thrashing, which scaling the number of cores wouldn’t fix.

The system may not have concurrency, in which case it cannot make use of the other cores to resolve the bottleneck.

Biii) False – an IT system can just have 2 resources: a single bottleneck and another resource that dominates it in all service classes.

Biv) False – They typically degrade performance and can make bottlenecks hard to identify, restricting the scalability of a system.