Serverless Shuffle Sharding

Improving resiliency and fault isolation in a multi-tenant SaaS environment

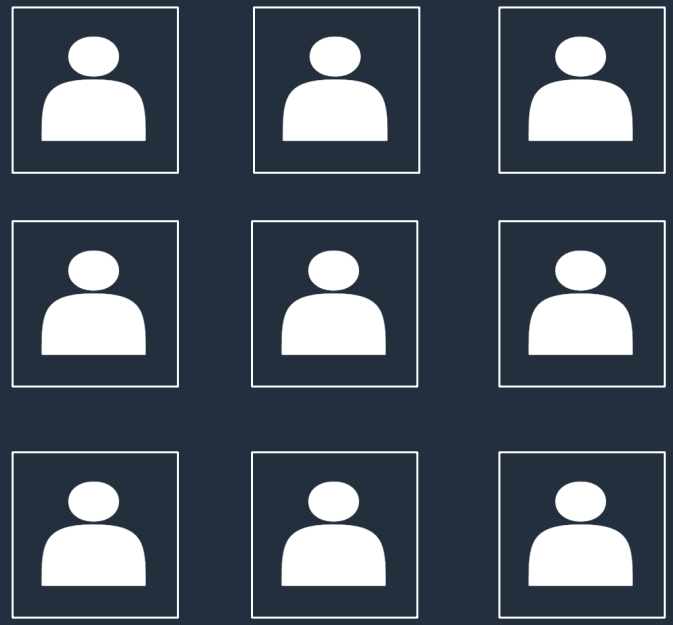
Anton Aleksandrov
Principal Solution Architect, Serverless
AWS

Reliability

The reliability pillar focuses on workloads performing their intended functions and how to recover quickly from failure to meet demands. Key topics include distributed system design, recovery planning, and adapting to changing requirements

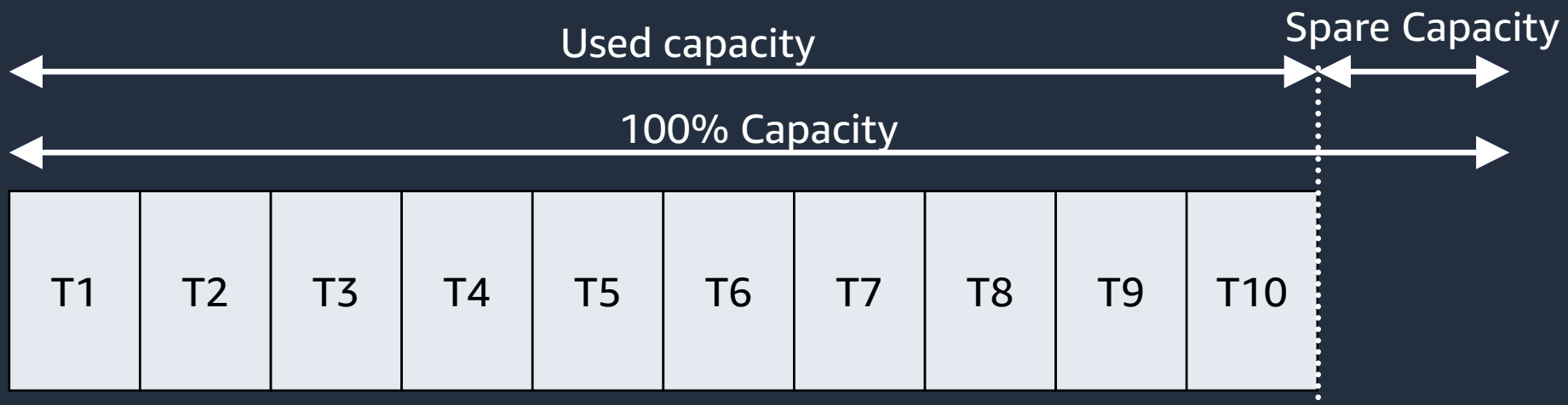
AWS Well-Architected Framework
https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html

What is a multi-tenant SaaS environment



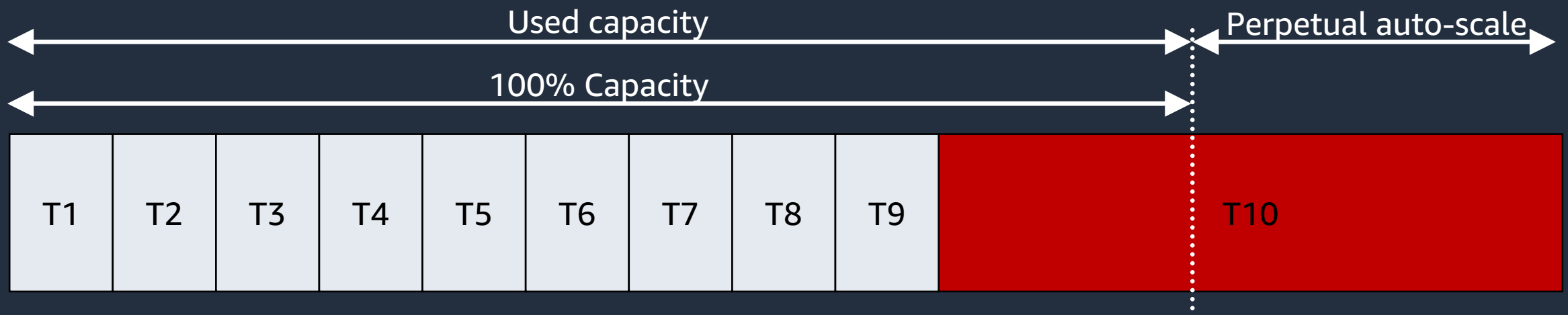
What is a noisy neighbors problem

Noisy neighbors



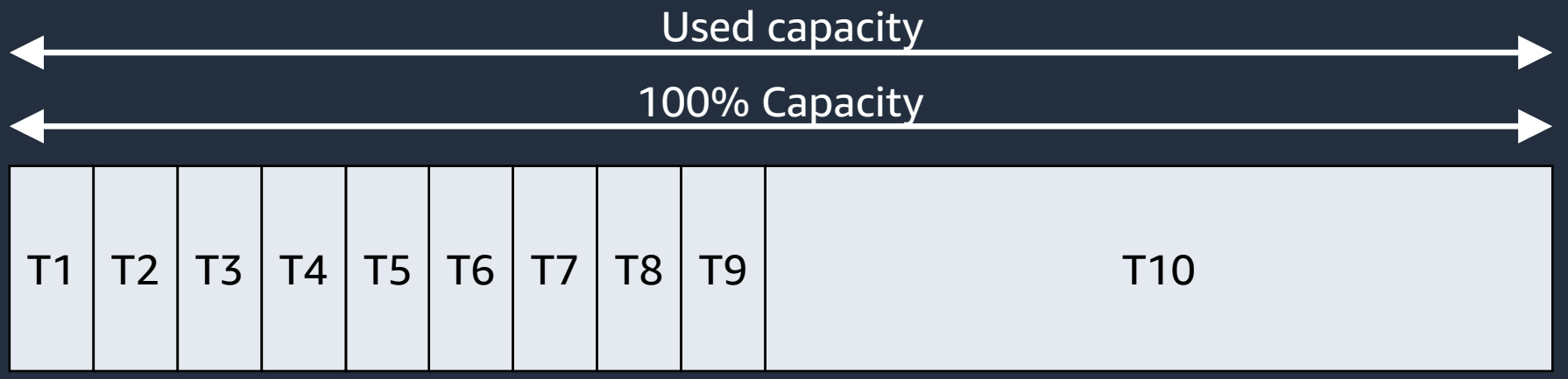
Utopia

Noisy neighbors



Expectation

Noisy neighbors

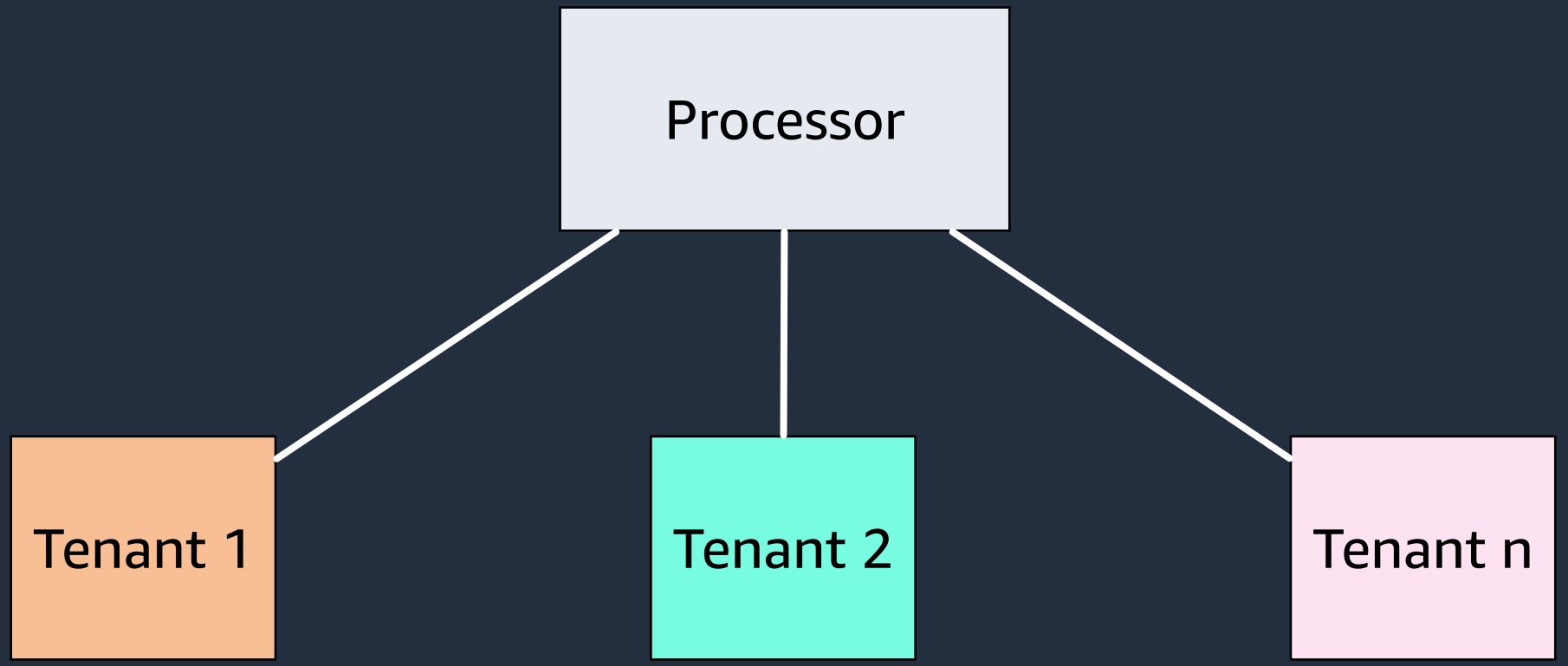


Reality

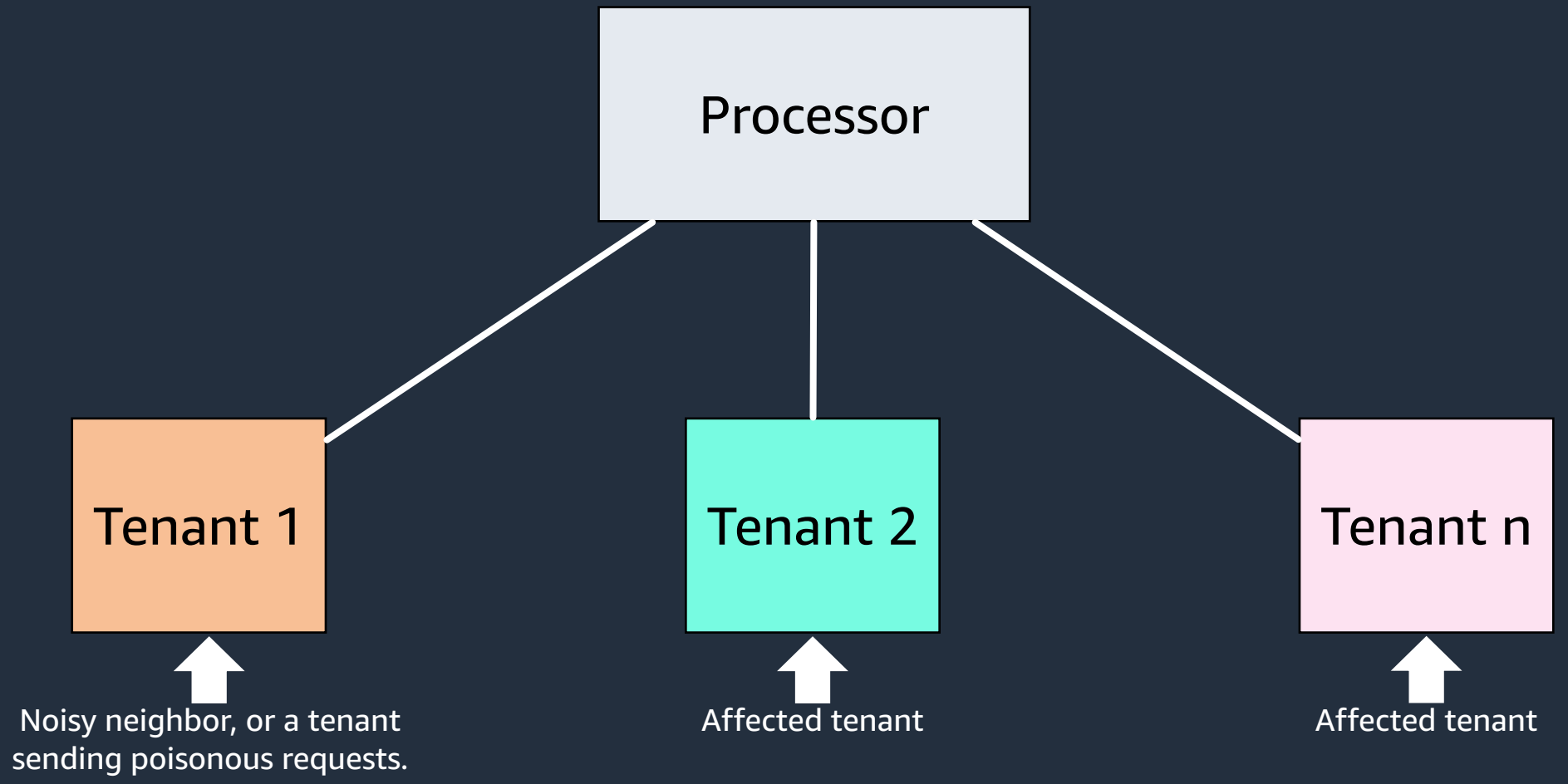
https://en.wikipedia.org/wiki/Cloud_computing_issues#Performance_interference_and_noisy_neighbors

Let’s see this in action

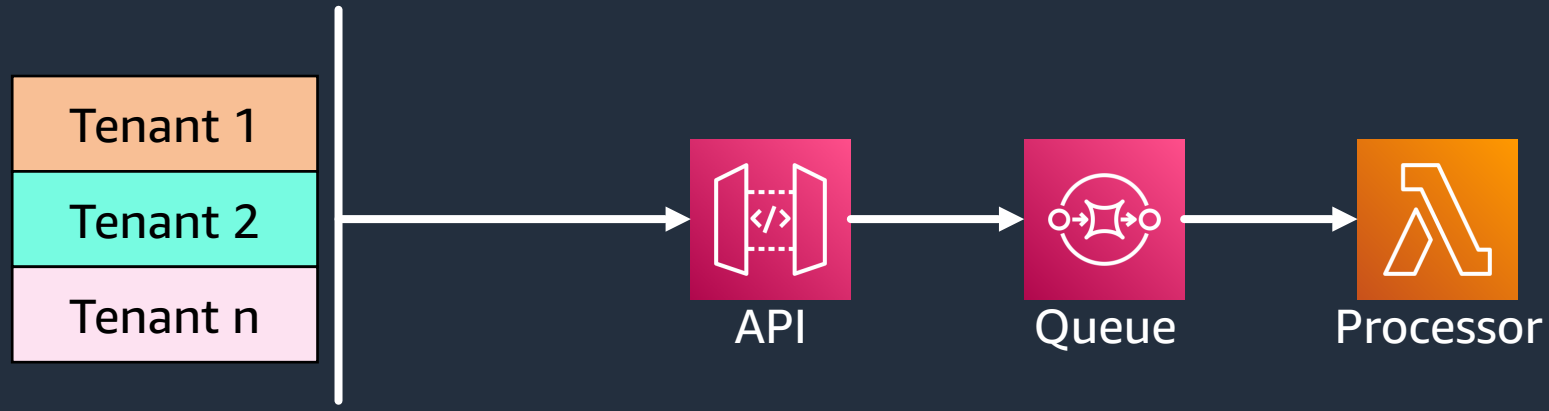
Single processor



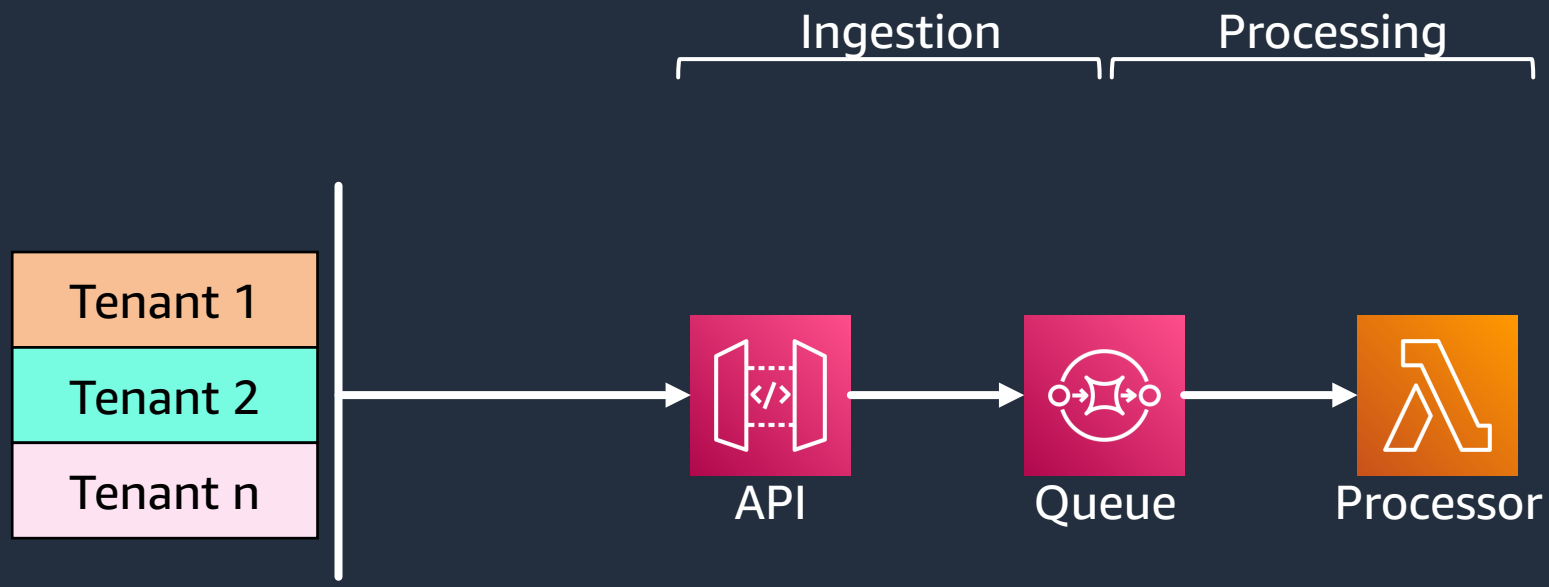
Single processor



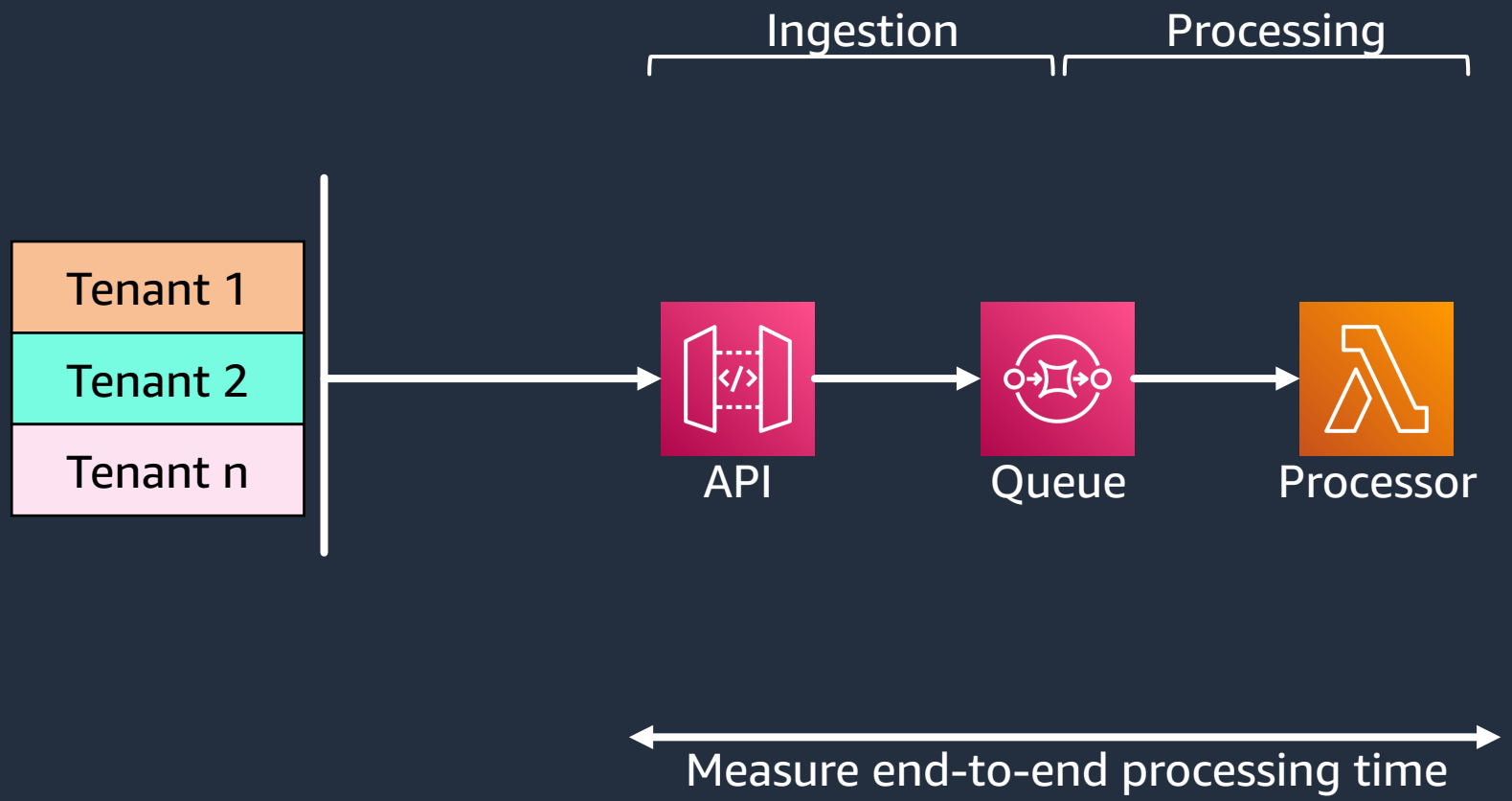
Single processor



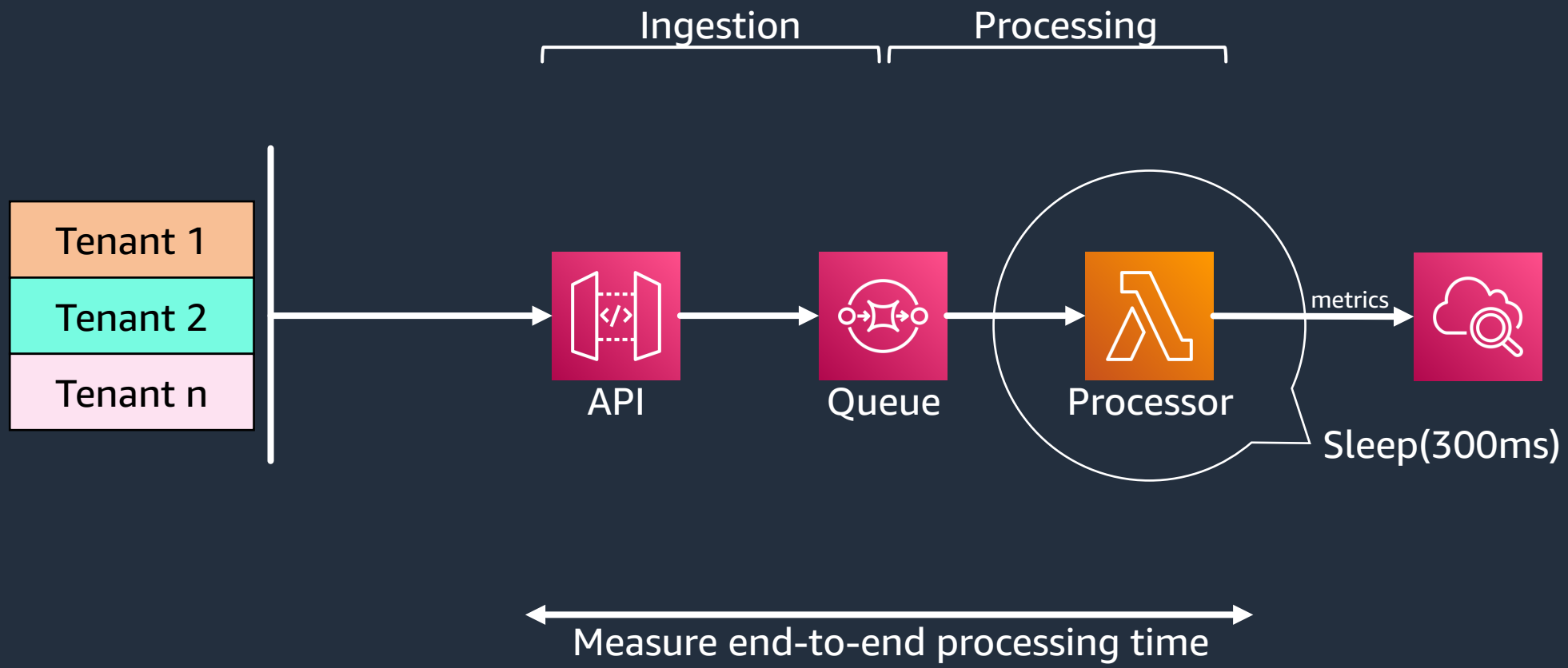
Single processor



Single processor

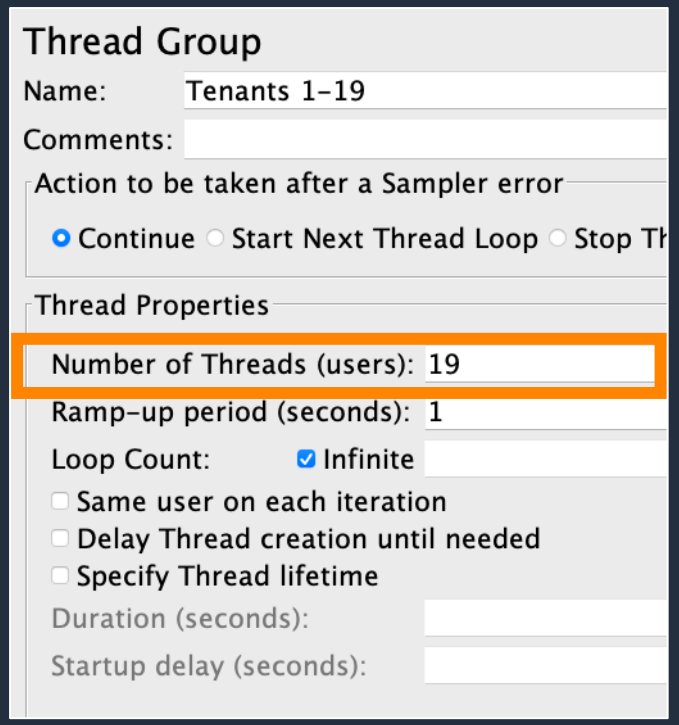


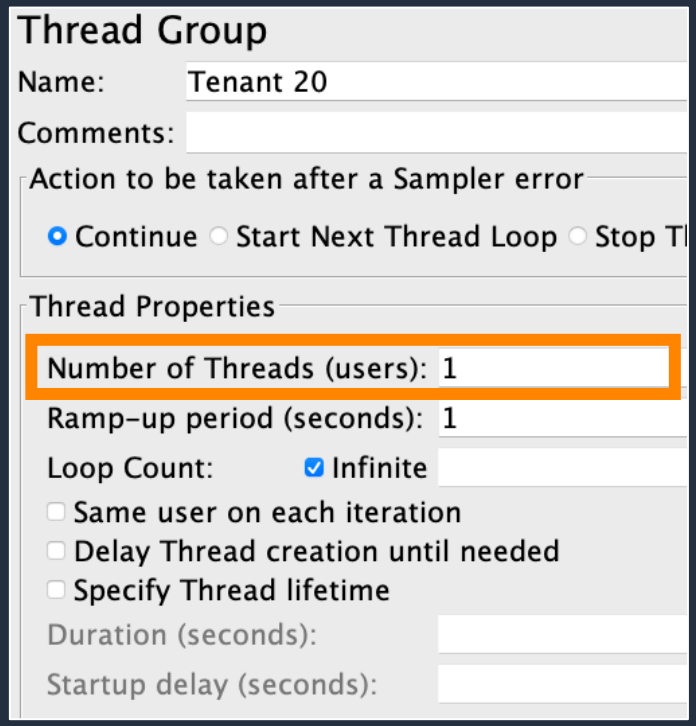
Single processor



Test setup

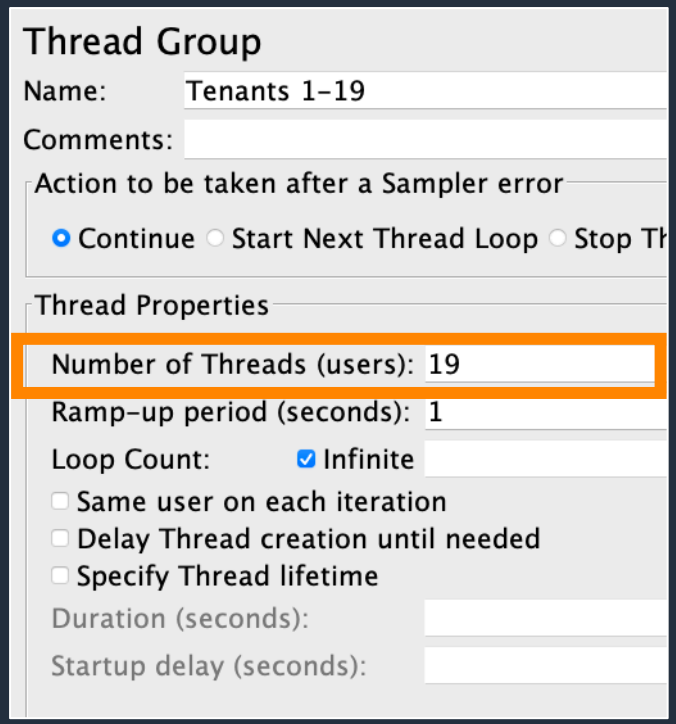
Step 1 – Peace and quiet

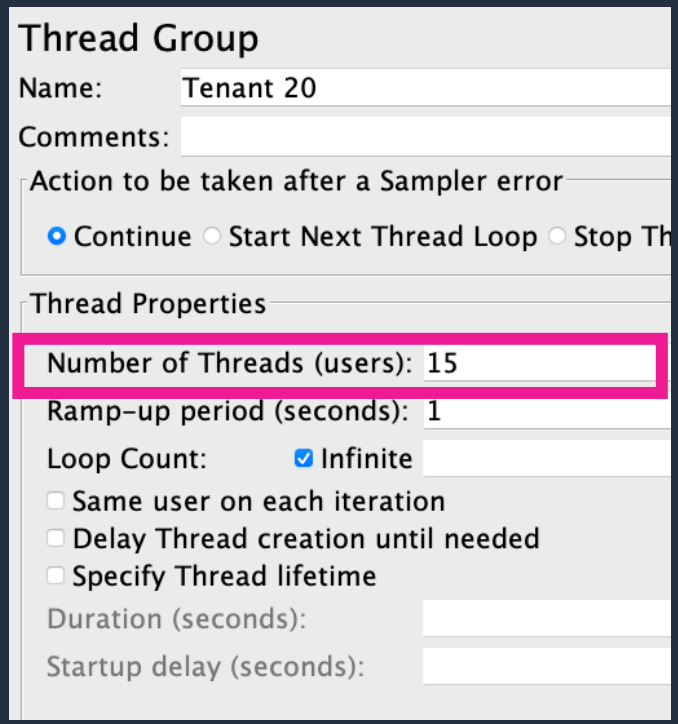




Test setup

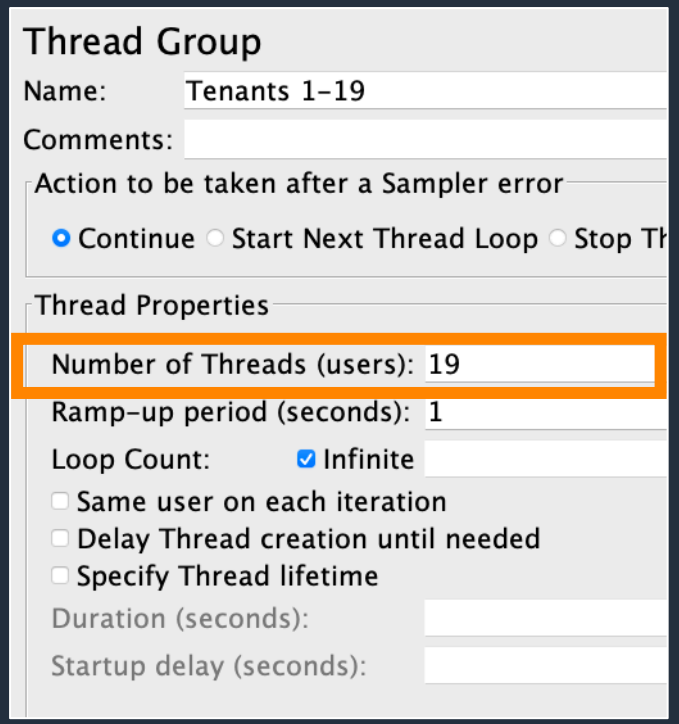
Step 2 – Noisy neighbor!

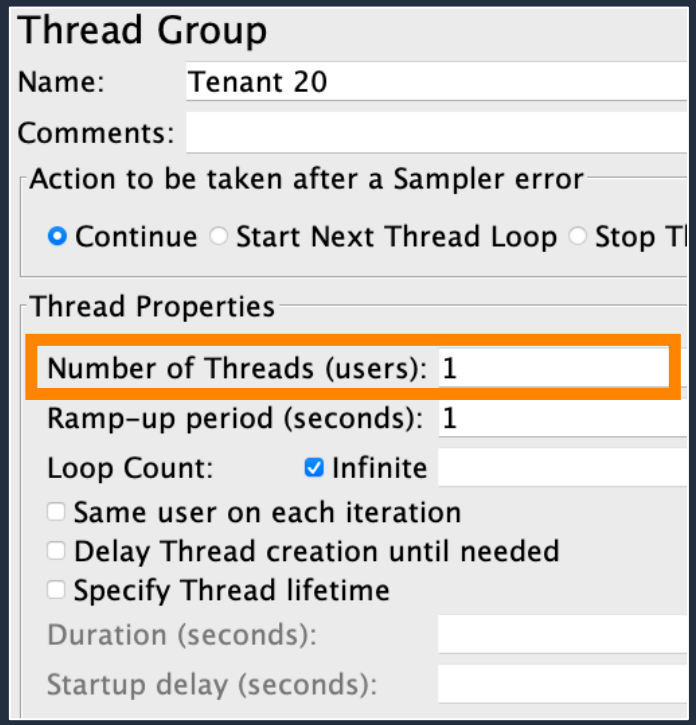




Test setup

Step 3 – Cooldown and recovery





Step 1 - Peace and quiet

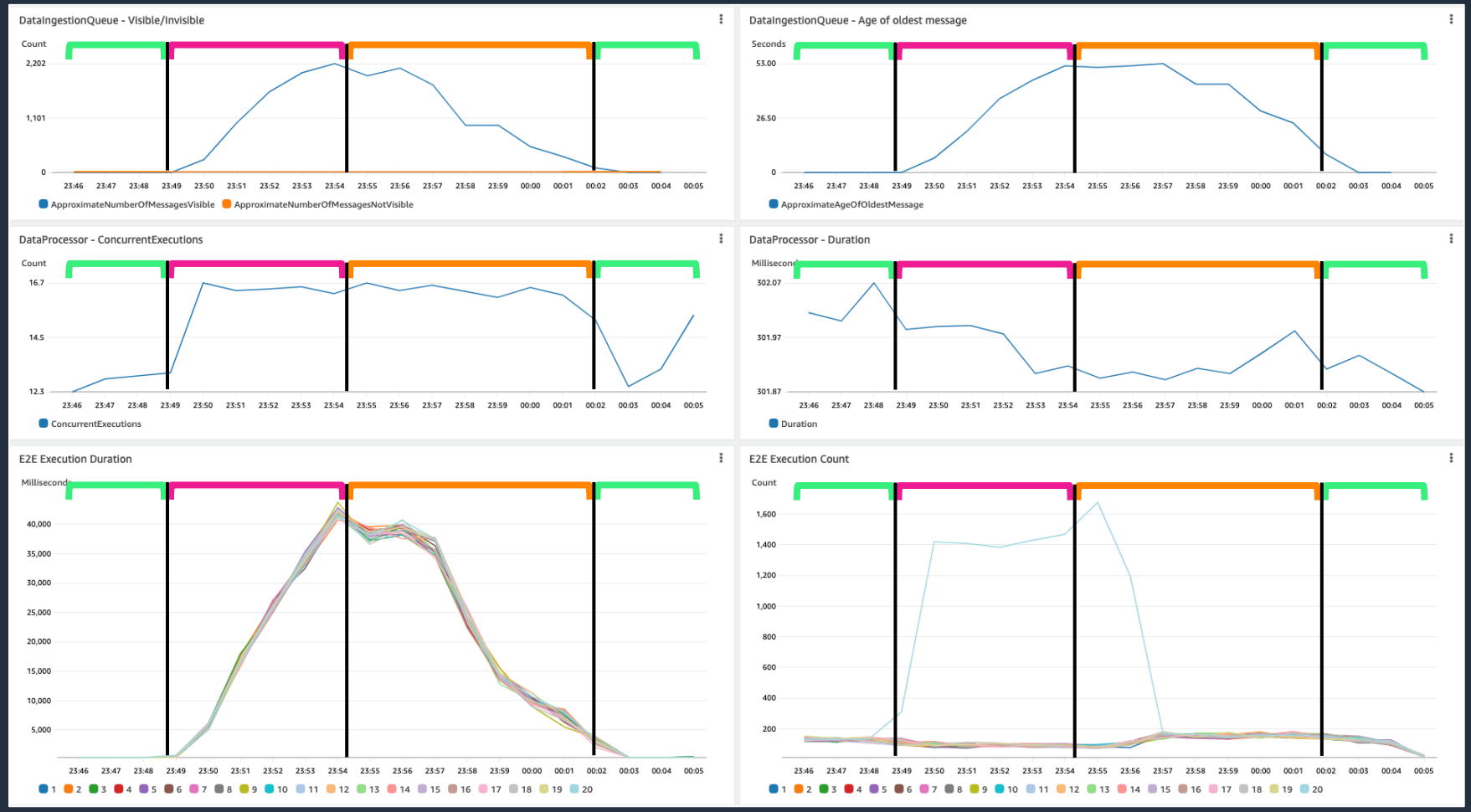
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	R
Request - Tenant 1	683	325	282	458	554	669	237	1216	0.00%	2.4/sec	
Request - Tenant 10	644	322	281	461	548	688	223	852	0.00%	2.2/sec	
Request - Tenant 11	666	322	282	452	540	741	238	1378	0.00%	2.3/sec	
Request - Tenant 12	678	321	283	456	539	606	235	1129	0.00%	2.3/sec	
Request - Tenant 13	657	325	284	453	558	783	233	990	0.00%	2.3/sec	
Request - Tenant 14	724	324	282	455	553	727	232	1327	0.00%	2.5/sec	
Request - Tenant 15	711	318	282	426	529	664	230	931	0.00%	2.5/sec	
Request - Tenant 16	659	319	281	454	542	646	232	945	0.00%	2.3/sec	
Request - Tenant 17	693	325	282	456	554	743	229	982	0.00%	2.4/sec	
Request - Tenant 18	687	323	282	454	556	673	237	1138	0.00%	2.4/sec	
Request - Tenant 19	697	329	285	504	553	619	235	1283	0.00%	2.4/sec	
Request - Tenant 2	691	325	284	456	544	684	226	968	0.00%	2.4/sec	
Request - Tenant 20	679	324	282	455	547	701	236	1129	0.00%	2.3/sec	
Request - Tenant 3	658	327	283	465	550	648	234	1158	0.00%	2.3/sec	
Request - Tenant 4	675	319	283	438	529	612	231	1261	0.00%	2.3/sec	
Request - Tenant 5	679	323	284	453	542	630	233	1273	0.00%	2.4/sec	
Request - Tenant 6	699	319	283	426	534	632	234	1254	0.00%	2.4/sec	
Request - Tenant 7	674	321	283	461	540	657	238	900	0.00%	2.3/sec	
Request - Tenant 8	706	325	282	458	551	760	234	1379	0.00%	2.4/sec	
Request - Tenant 9	639	316	280	446	518	666	235	936	0.00%	2.2/sec	
TOTAL	13599	323	283	456	546	688	223	1379	0.00%	46.9/sec	

Step 2 – Noisy neighbor!

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput
Request – Tenant 1	673	406	315	589	661	990	221	1282	0.00%	2.0/sec
Request – Tenant 10	653	402	311	580	665	864	225	1143	0.00%	1.9/sec
Request – Tenant 11	680	404	294	589	720	1084	228	1281	0.00%	2.0/sec
Request – Tenant 12	643	404	304	586	657	1032	232	1277	0.00%	1.9/sec
Request – Tenant 13	646	411	306	615	729	1043	230	1118	0.00%	1.9/sec
Request – Tenant 14	629	404	297	597	678	920	229	1213	0.00%	1.9/sec
Request – Tenant 15	664	393	291	578	654	865	222	1226	0.00%	2.0/sec
Request – Tenant 16	672	410	324	596	685	1011	227	1208	0.00%	2.0/sec
Request – Tenant 17	642	397	287	588	674	1051	228	1250	0.00%	1.9/sec
Request – Tenant 18	689	397	293	579	673	985	224	1141	0.00%	2.0/sec
Request – Tenant 19	655	405	293	590	711	1012	231	1150	0.00%	1.9/sec
Request – Tenant 20	9968	408	310	590	691	1060	220	1579	0.00%	29.3/sec
Request – Tenant 4	657	415	339	595	700	1051	231	1174	0.00%	2.0/sec
Request – Tenant 5	701	400	297	582	657	966	227	1198	0.00%	2.1/sec
Request – Tenant 6	723	403	294	596	673	1044	231	2183	0.00%	2.1/sec
Request – Tenant 7	669	408	304	590	719	1051	231	1294	0.00%	2.0/sec
Request – Tenant 8	692	401	305	588	671	920	233	1168	0.00%	2.0/sec
Request – Tenant 9	710	407	289	597	689	1045	225	1661	0.00%	2.1/sec
TOTAL	22690	405	304	590	690	1039	220	2183	0.00%	66.7/sec

Results

Results

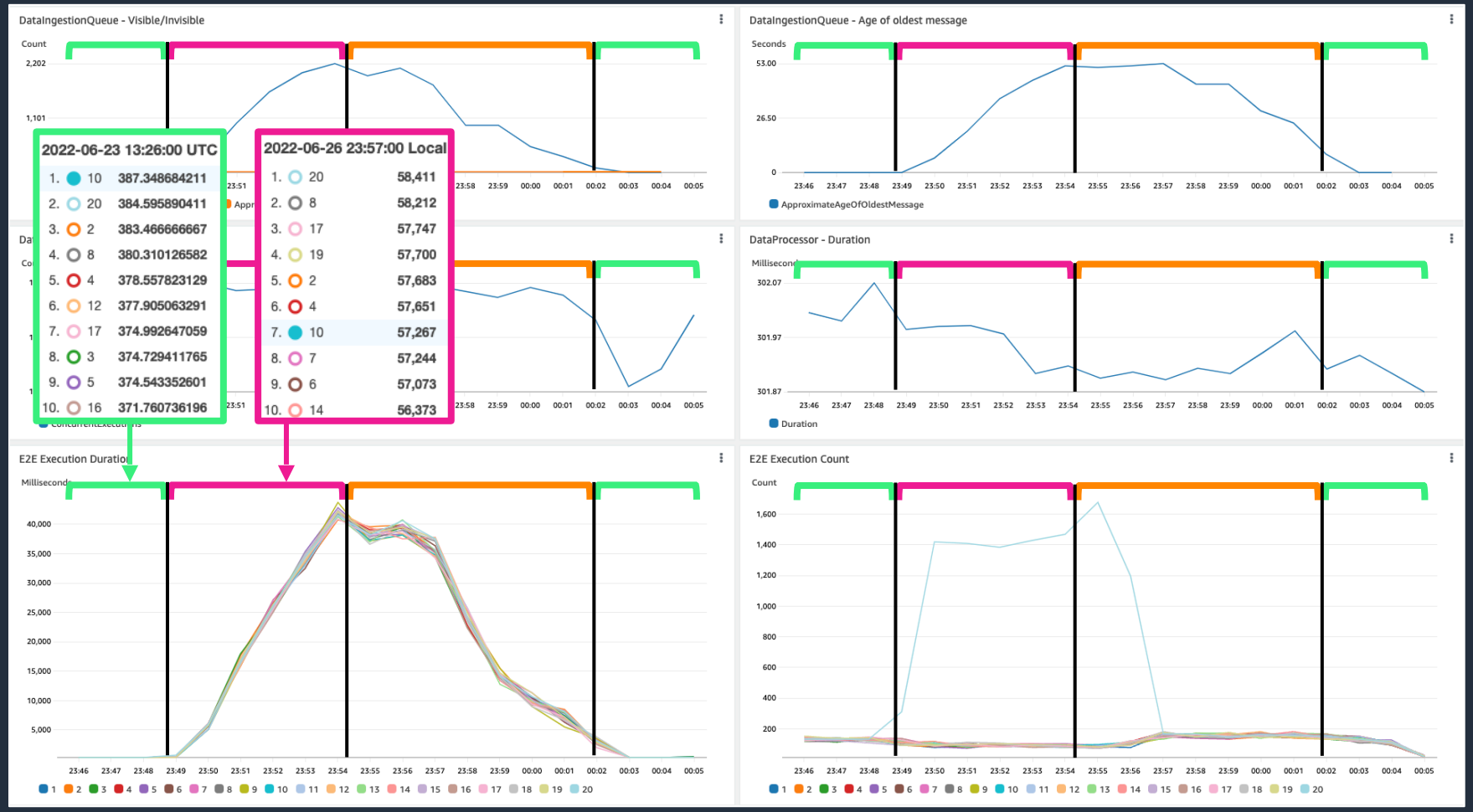


Peace and quiet

Noisy neighbor

Cooldown

Results



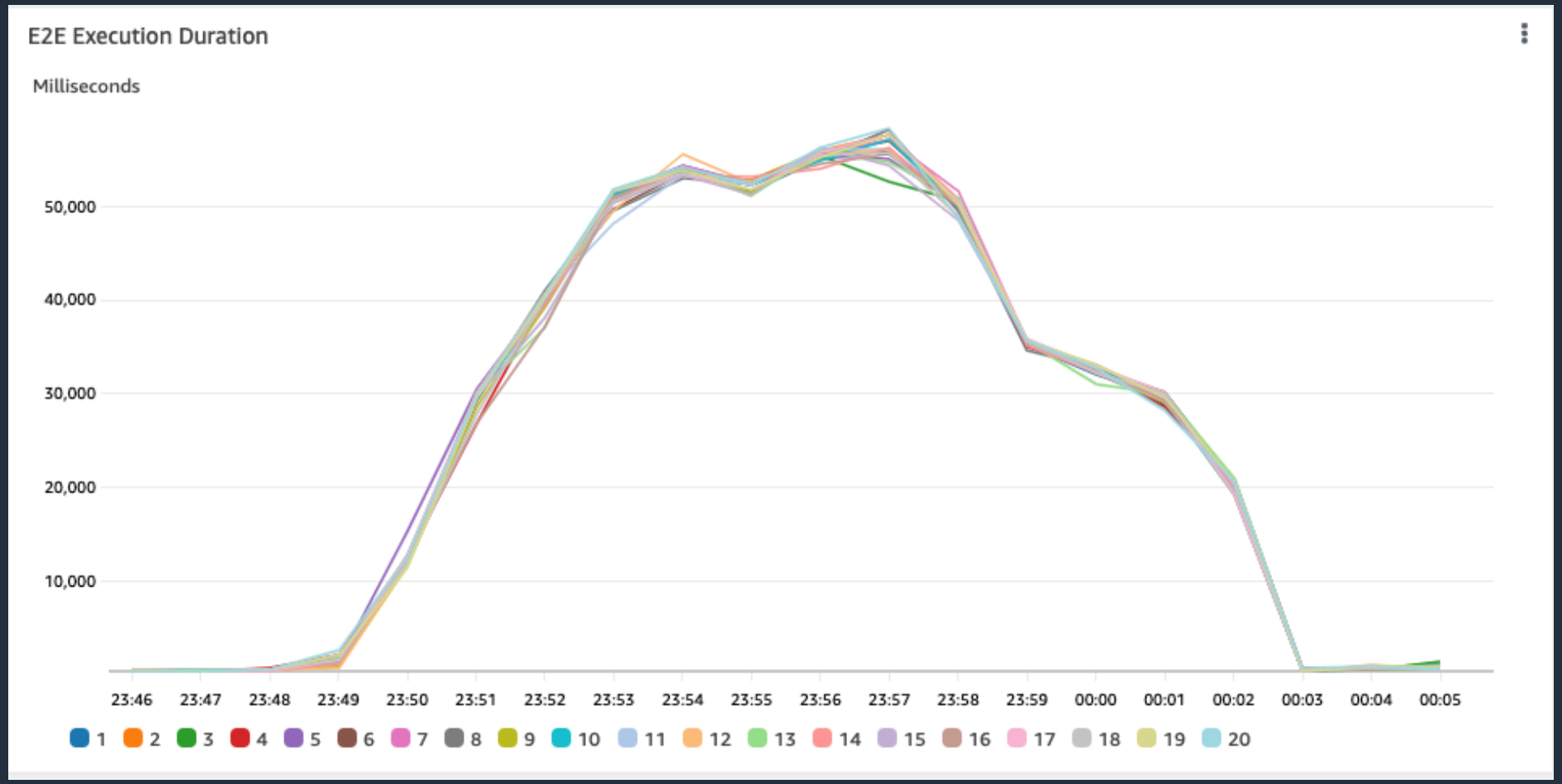
Peace and quiet

Noisy neighbor

Cooldown

Results – P99

P99

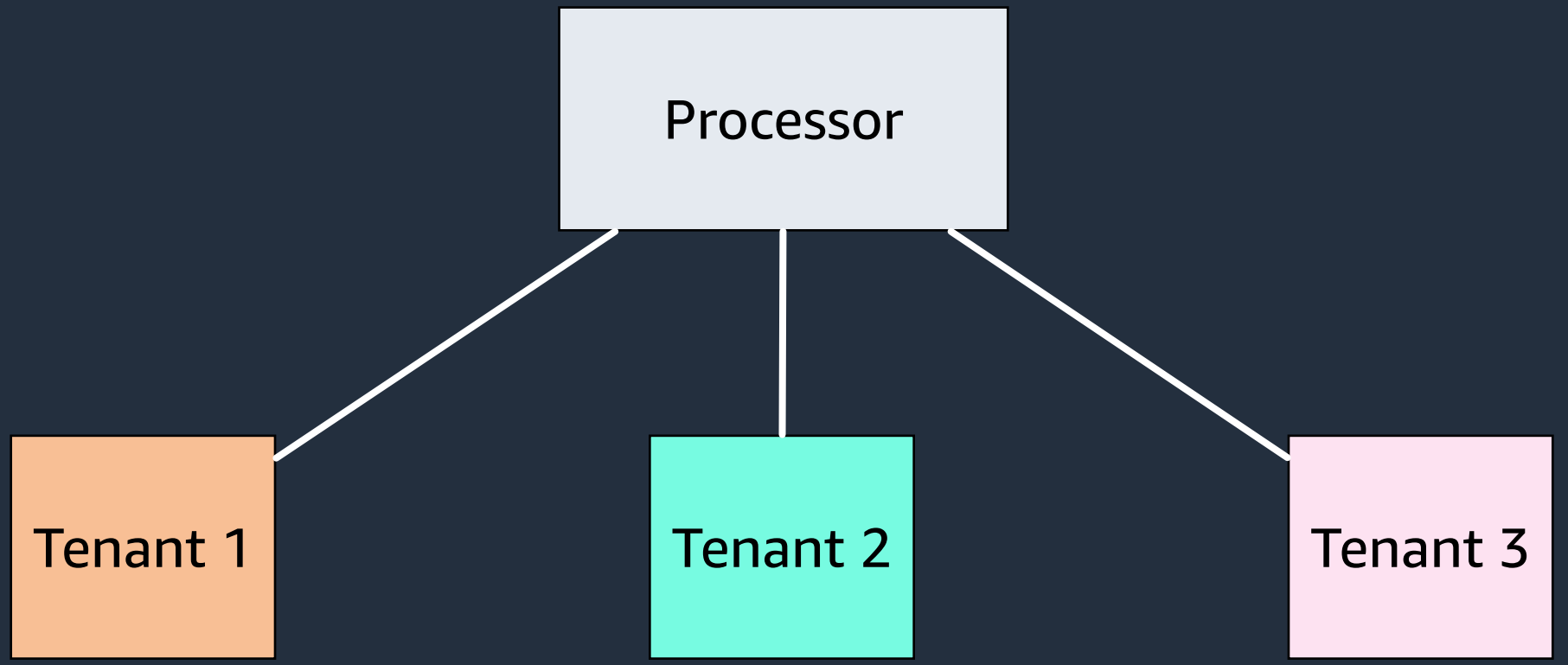


Conclusions

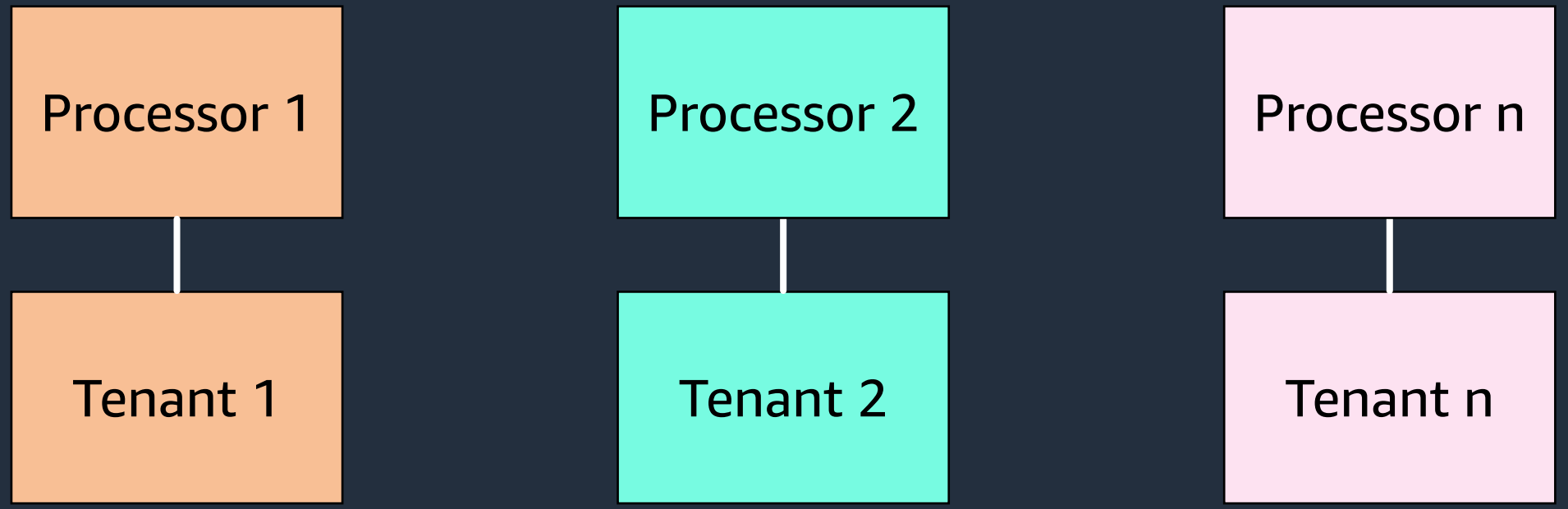
- A single noisy neighbor or a tenant sending “poisonous” requests can have adverse effect on every other tenant
- Building decoupled asynchronous systems with messaging services like SQS helps with vertical fault isolation
- Use API Gateway throttling mechanism where applicable
- Cooldown depends on the volume of spare capacity
- Consider fault isolation techniques when building multi-tenant SaaS systems

Brainstorming a solution

Single processor

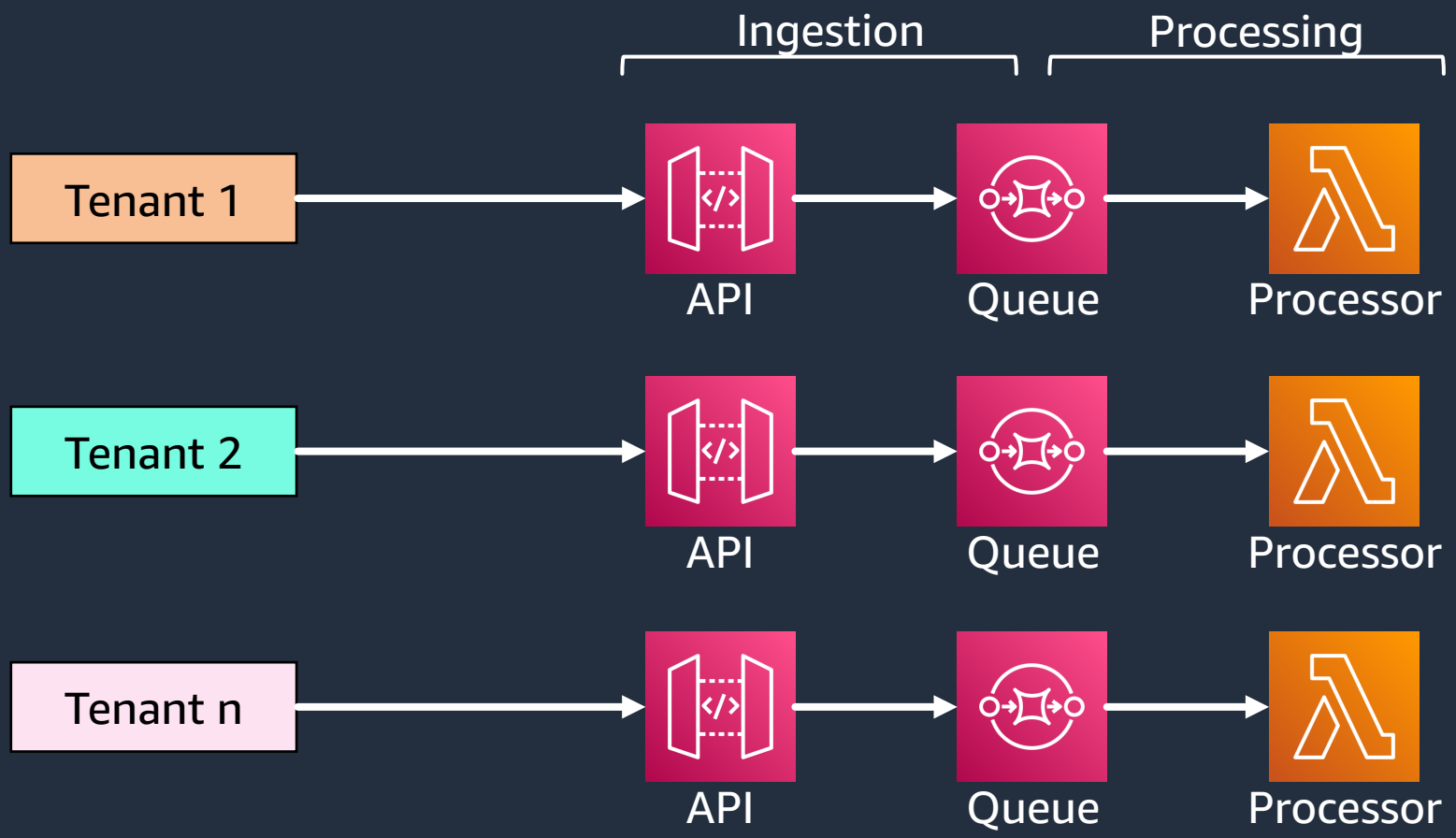


Processor per tenant

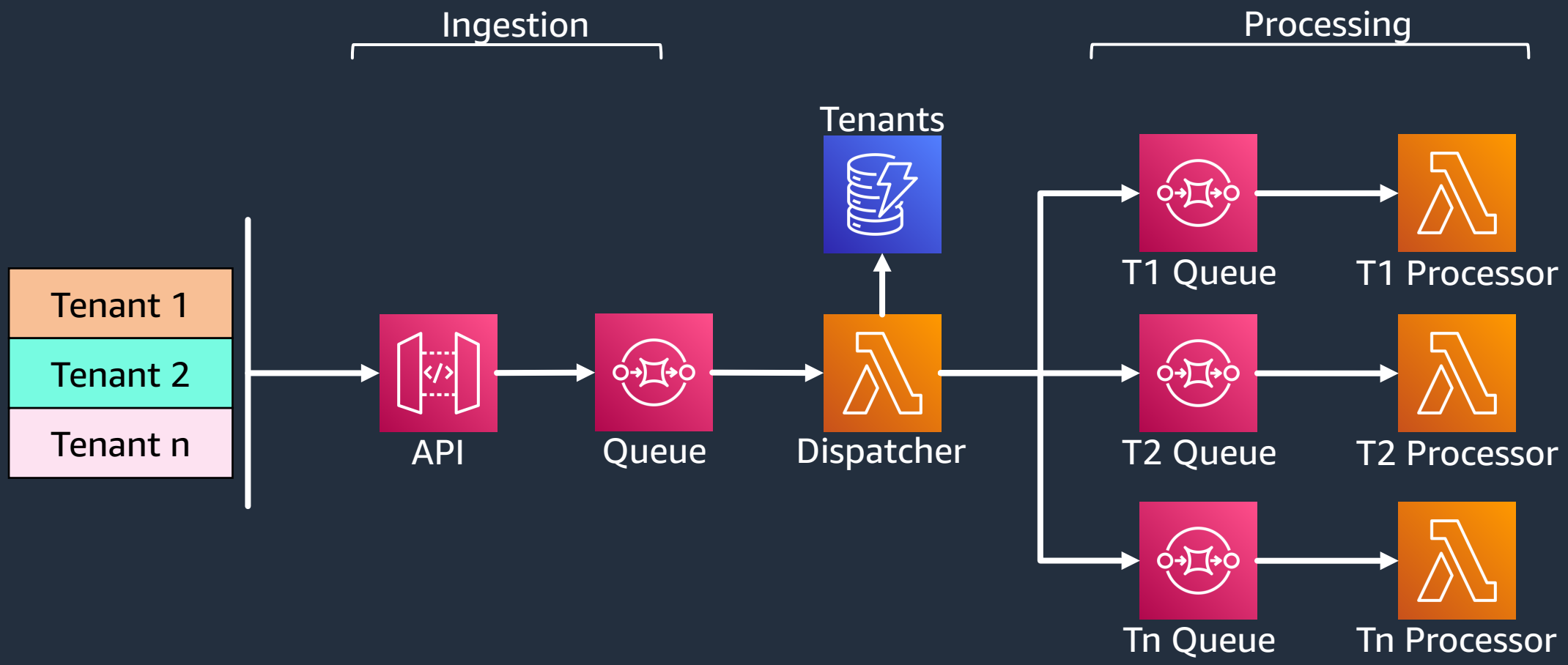


P==T

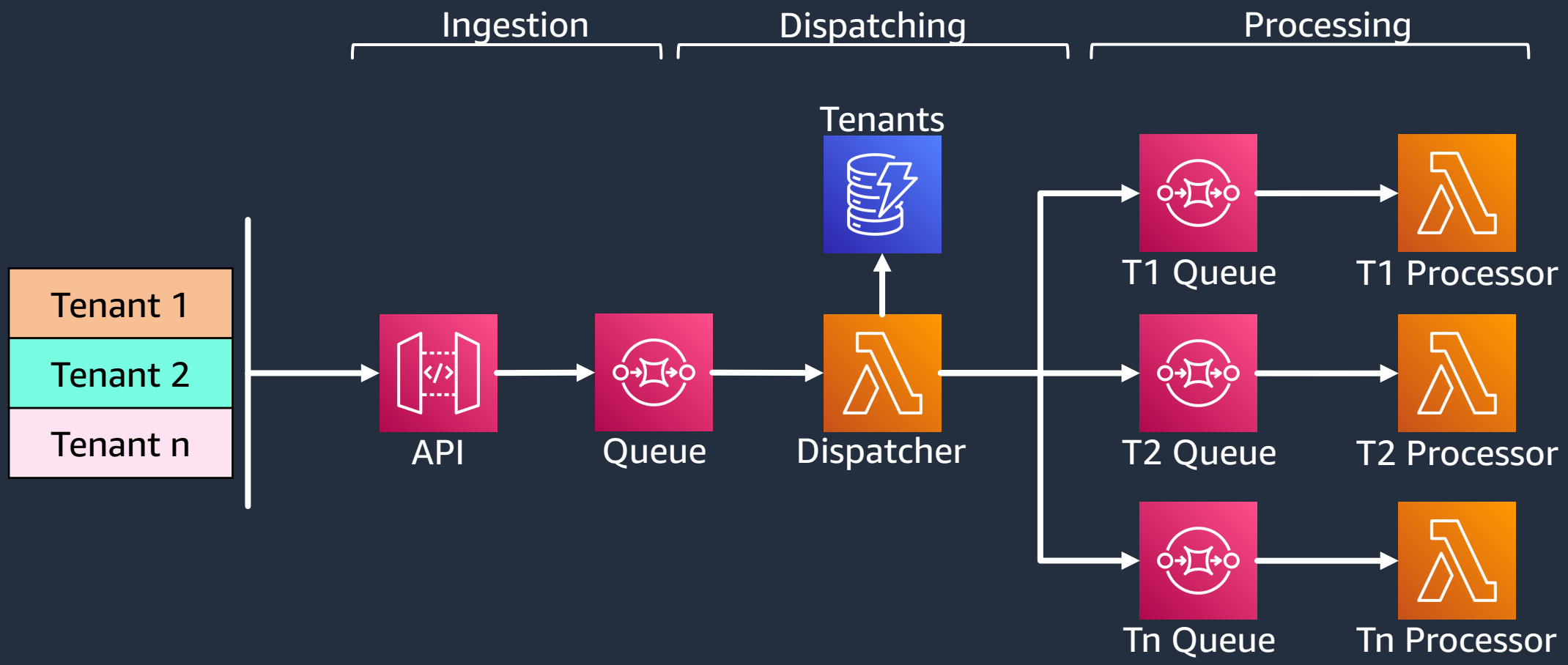
Processor per tenant – sharing nothing



Processor per tenant – sharing ingestion



Processor per tenant – sharing ingestion



Comparing the options

P == 1

One processor for all tenants

$

Vertical scaling

Major noisy neighbor problem

P << T

Number of processors is significantly lower than number of tenants

$$

Sublinear scaling (good!)

But would it address the noisy neighbor problem?

P==T

Number of processors equals to number of tenants

$$$$

Linear scaling (bad!)

No noisy neighbor problem!

Sharding 101

ID	Name	Stock	Price
1	Refrigerator	75	$1800
2	Dish washer	102	$700
3	TV	39	$1500
4	Camera	55	$400

Vertical Shard 1

ID	Name	Price
1	Refrigerator	$1800
2	Dish washer	$700
3	TV	$1500
4	Camera	$400

Vertical Shard 2

ID	Stock
1	75
2	102
3	39
4	55

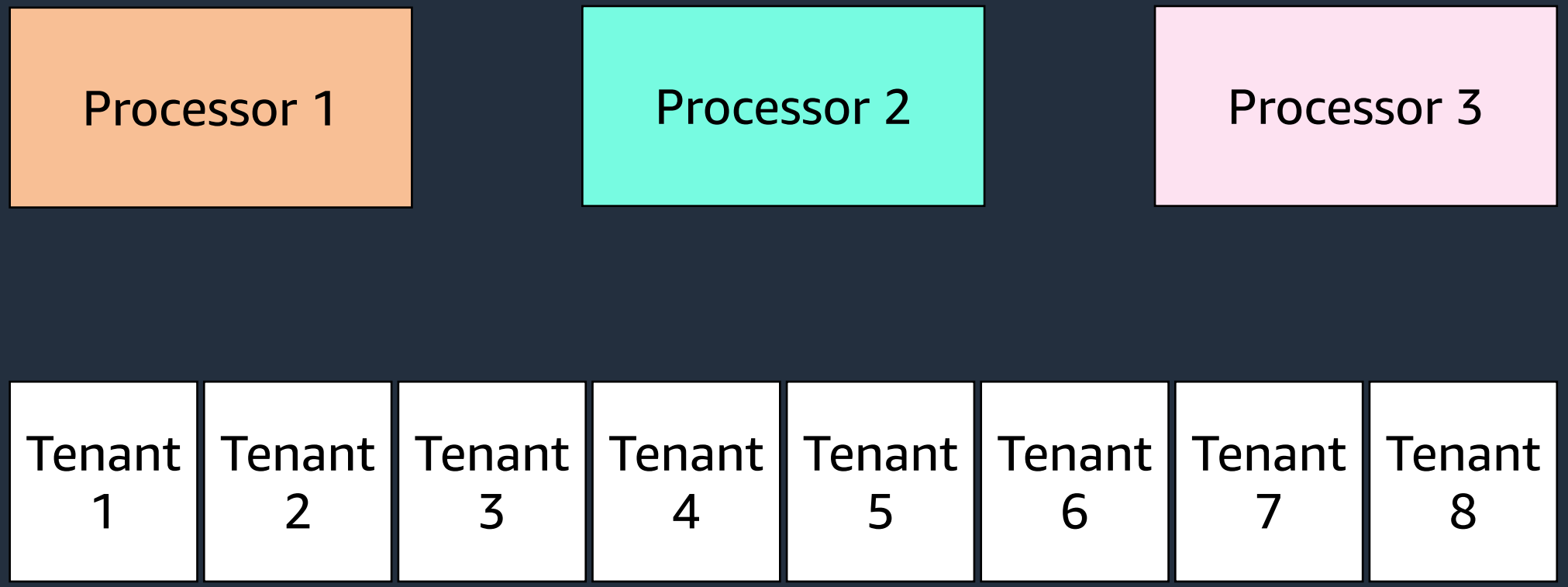
Horizontal Shard 1

ID	Name	Stock	Price
1	Refrigerator	75	$1800
2	Dish washer	102	$700

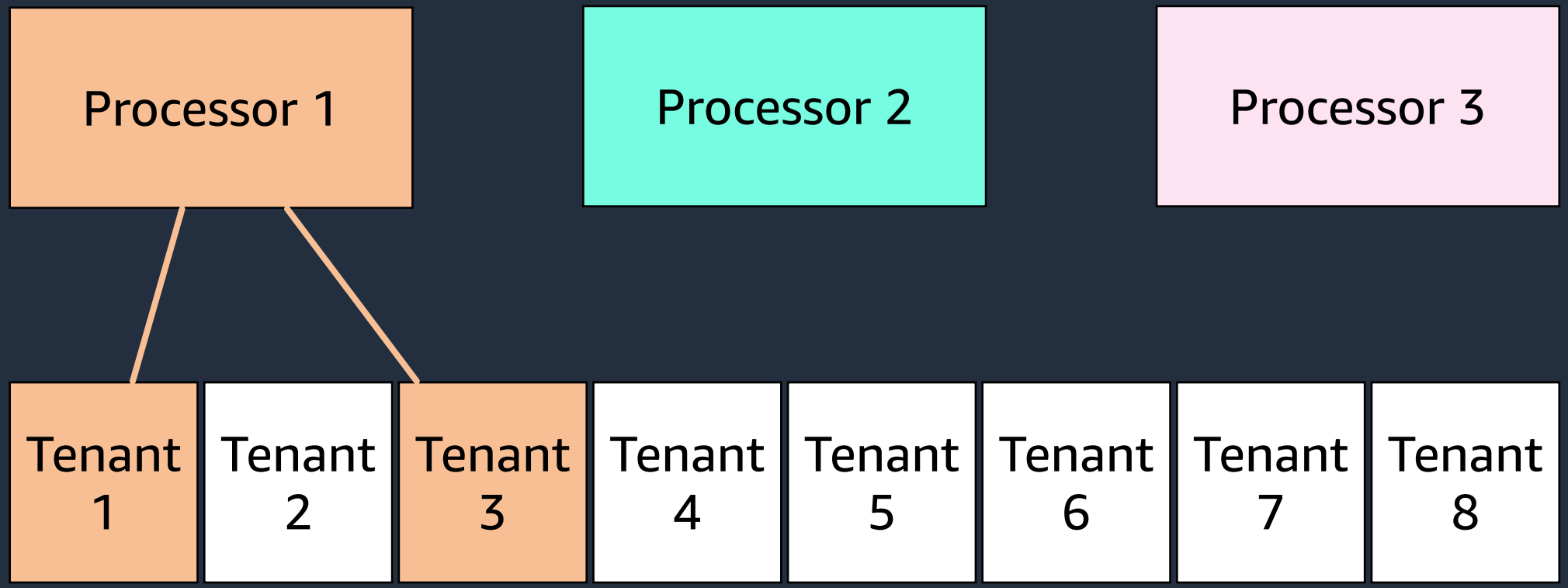
Horizontal Shard 2

ID	Name	Stock	Price
3	TV	39	$1500
4	Camera	55	$400

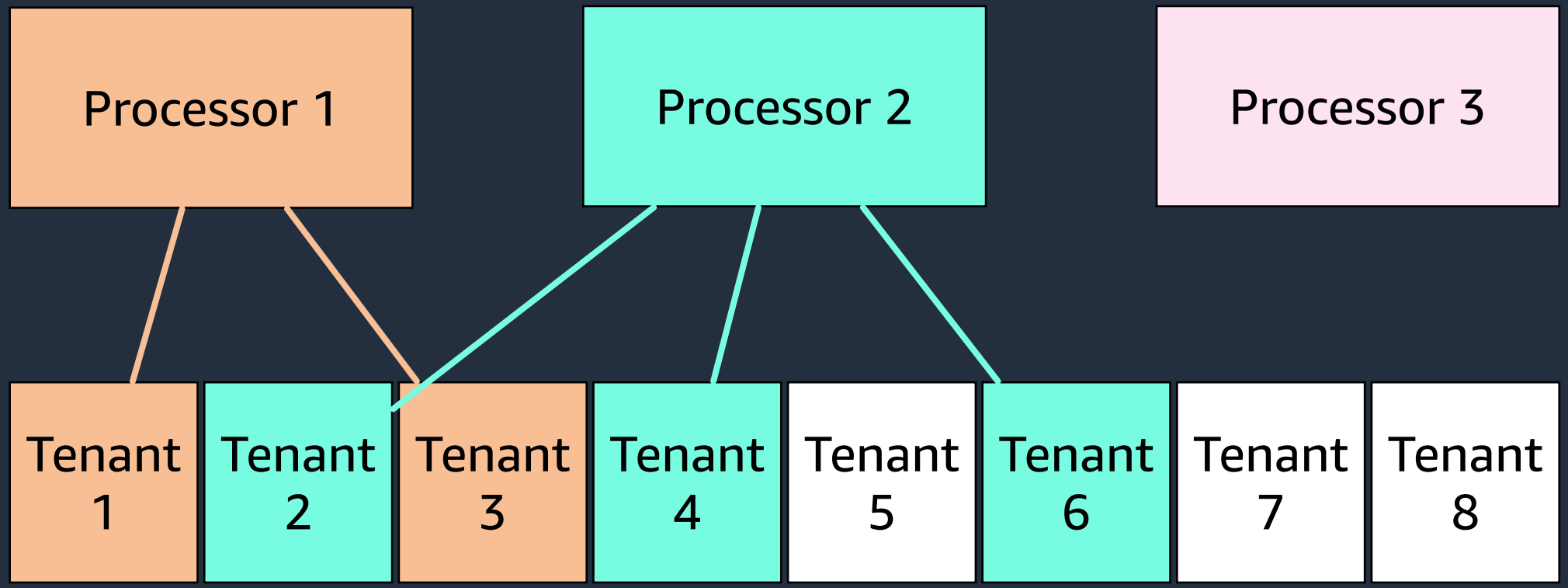
Simple sharding



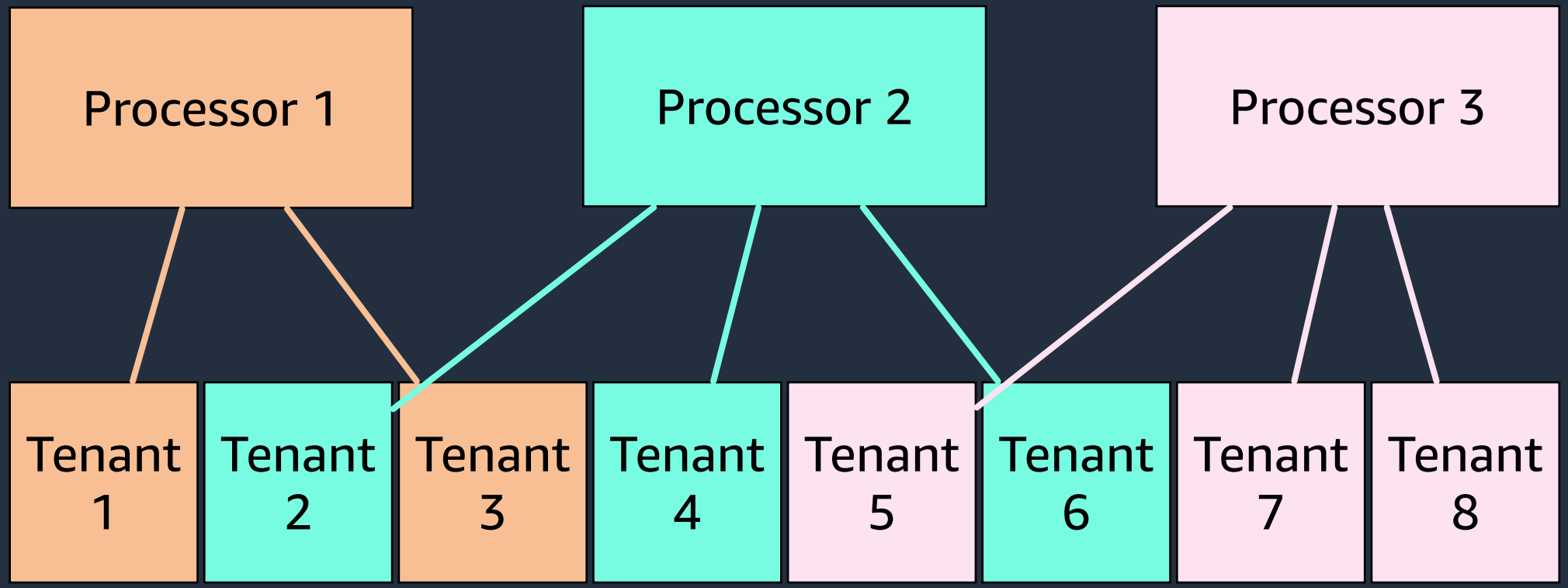
Simple sharding



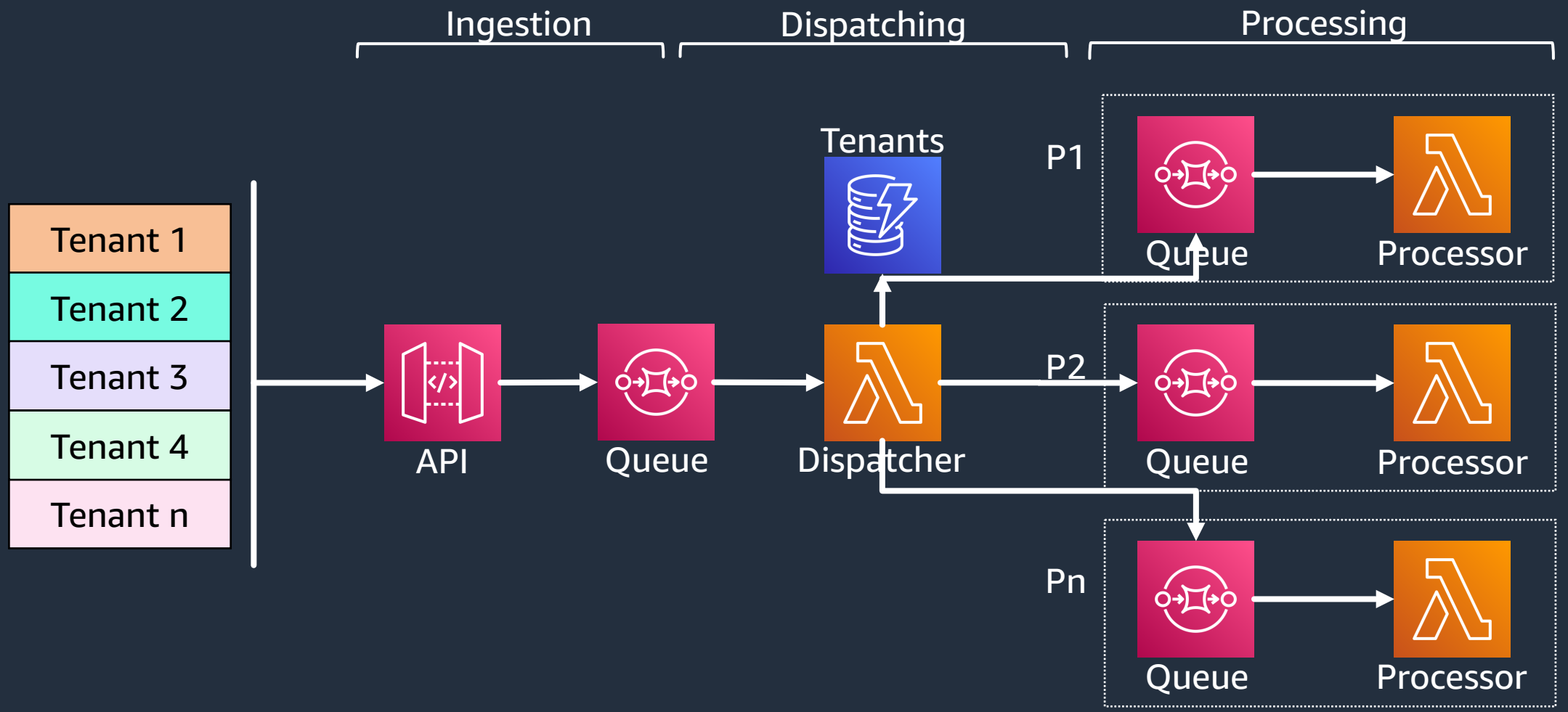
Simple sharding



Simple sharding



Simple sharding: P << T



Let’s see this in action

Noisy neighbor!

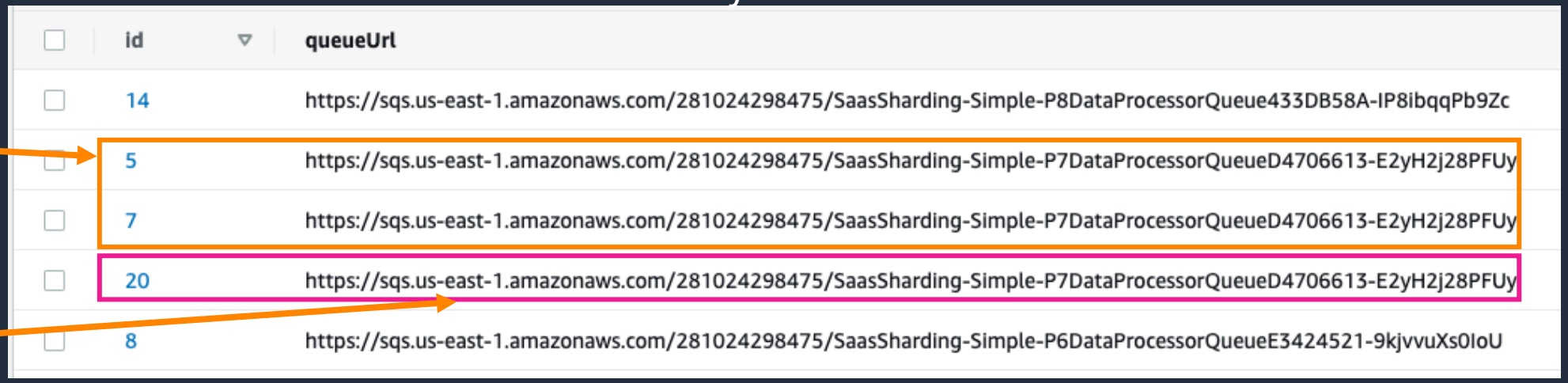
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput
Request - Tenant 1	673	406	315	589	661	990	221	1282	0.00%	2.0/sec
Request - Tenant 10	653	402	311	580	665	864	225	1143	0.00%	1.9/sec
Request - Tenant 11	680	404	294	589	720	1084	228	1281	0.00%	2.0/sec
Request - Tenant 12	643	404	304	586	657	1032	232	1277	0.00%	1.9/sec
Request - Tenant 13	646	411	306	615	729	1043	230	1118	0.00%	1.9/sec
Request - Tenant 14	629	404	297	597	678	920	229	1213	0.00%	1.9/sec
Request - Tenant 15	664	393	291	578	654	865	222	1226	0.00%	2.0/sec
Request - Tenant 16	672	410	324	596	685	1011	227	1208	0.00%	2.0/sec
Request - Tenant 17	642	397	287	588	674	1051	228	1250	0.00%	1.9/sec
Request - Tenant 18	689	397	293	579	673	985	224	1141	0.00%	2.0/sec
Request - Tenant 19	655	405	293	590	711	1012	231	1150	0.00%	1.9/sec
Request - Tenant 2	661	413	344	662	705	921	230	1080	0.00%	1.9/sec
Request - Tenant 20	9968	408	310	590	691	1060	220	1579	0.00%	29.3/sec
Request - Tenant 3	669	392	297	585	698	997	229	1390	0.00%	2.0/sec
Request - Tenant 4	657	415	339	595	700	1051	231	1174	0.00%	2.0/sec
Request - Tenant 5	701	400	297	582	657	966	227	1198	0.00%	2.1/sec
Request - Tenant 6	723	403	294	596	673	1044	231	2183	0.00%	2.1/sec
Request - Tenant 7	669	408	304	590	719	1051	231	1294	0.00%	2.0/sec
Request - Tenant 8	692	401	305	588	671	920	233	1168	0.00%	2.0/sec
Request - Tenant 9	710	407	289	597	689	1045	225	1661	0.00%	2.1/sec
TOTAL	22690	405	304	590	690	1039	220	2183	0.00%	66.7/sec

Before we begin

Tenants Table - DynamoDB

Affected tenants

Noisy neighbor



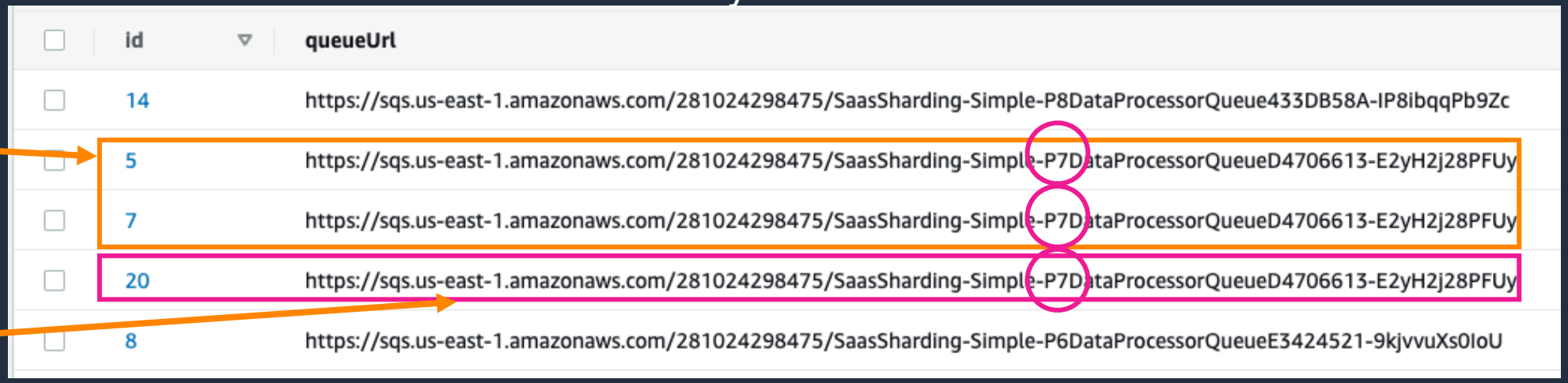
id	queueUrl
14	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P8DataProcessorQueue433DB58A-IP8ibqqPb9Zc
5	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
7	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
20	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
8	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P6DataProcessorQueueE3424521-9kjvvuXs0IoU

Before we begin

Tenants Table - DynamoDB

Affected tenants

Noisy neighbor



id	queueUrl
14	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P8DataProcessorQueue433DB58A-IP8ibqqPb9Zc
5	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
7	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
20	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
8	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P6DataProcessorQueueE3424521-9kjvvuXs0IoU

Before we begin

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

Tenants Table - DynamoDB

Affected tenants

Noisy neighbor

id	queueUrl
14	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P8DataProcessorQueue433DB58A-IP8ibqqPb9Zc
5	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
7	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
20	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P7DataProcessorQueueD4706613-E2yH2j28PFUy
8	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Simple-P6DataProcessorQueueE3424521-9kjvvuXs0IoU

Results

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

Results

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

Peace and quiet

Noisy neighbor

Cooldown

Results

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

Peace and quiet

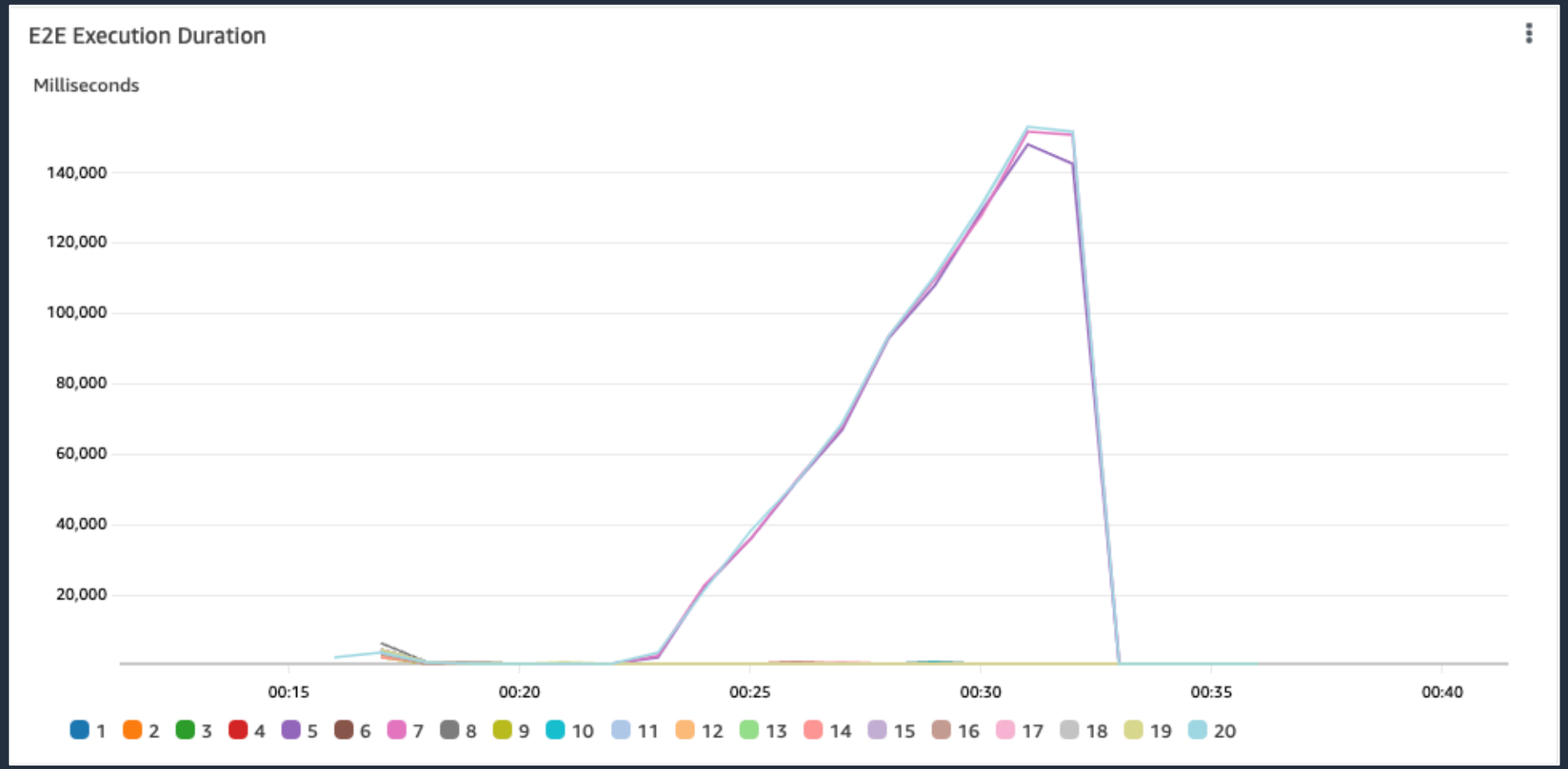
Noisy neighbor

Cooldown

Results – P99

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

P99



Results

Mental note
Noisy neighbor – 20
Affected tenants – 5,7
Affected processor – P7

Conclusions

$$\text{Number of affected tenants} = \text{roundup}\left(\frac{\text{number of tenants}}{\text{number of processors}}\right) = \frac{20}{8}$$

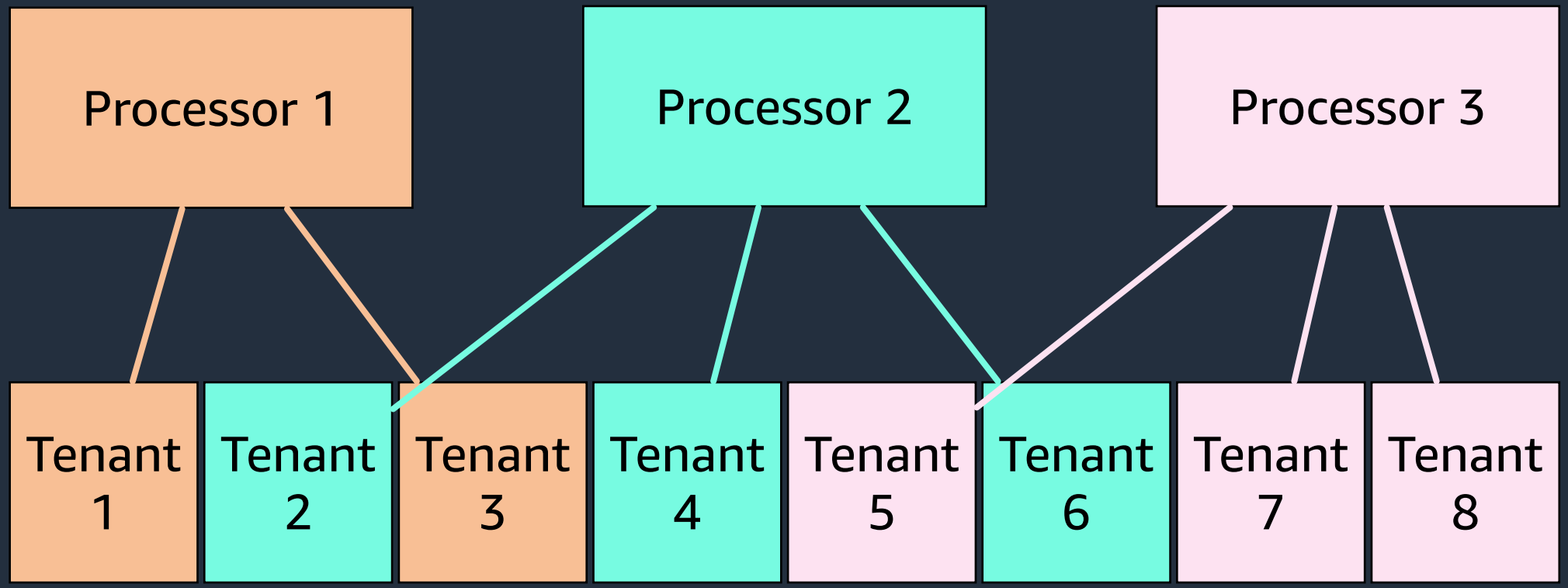
$$= 3$$

$$\text{Blast radius} = \frac{\text{number of affected tenants}}{\text{total number of tenants}} = \frac{3}{20} = 15\%$$

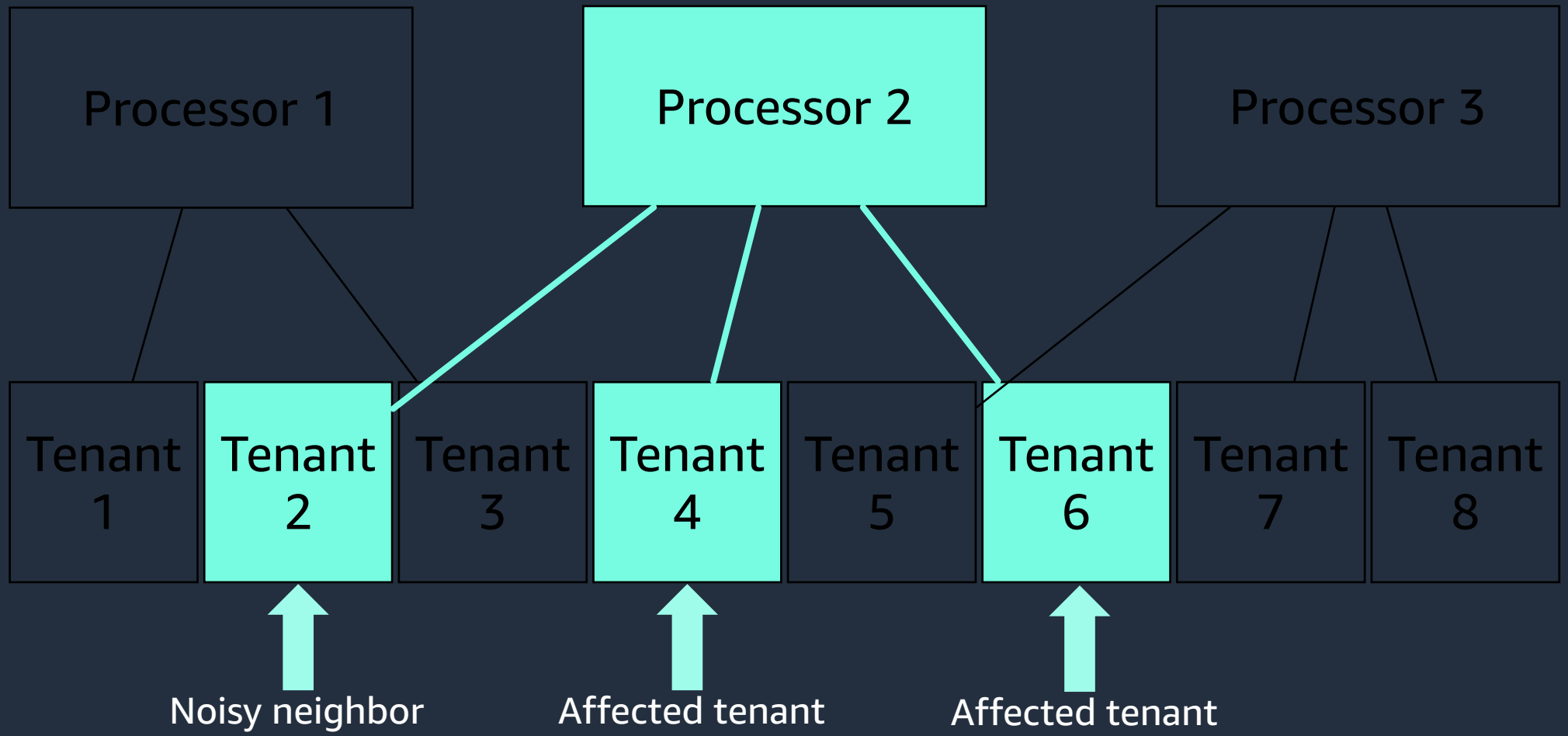
Not bad. But can we do even better?!

Let’s brainstorm further

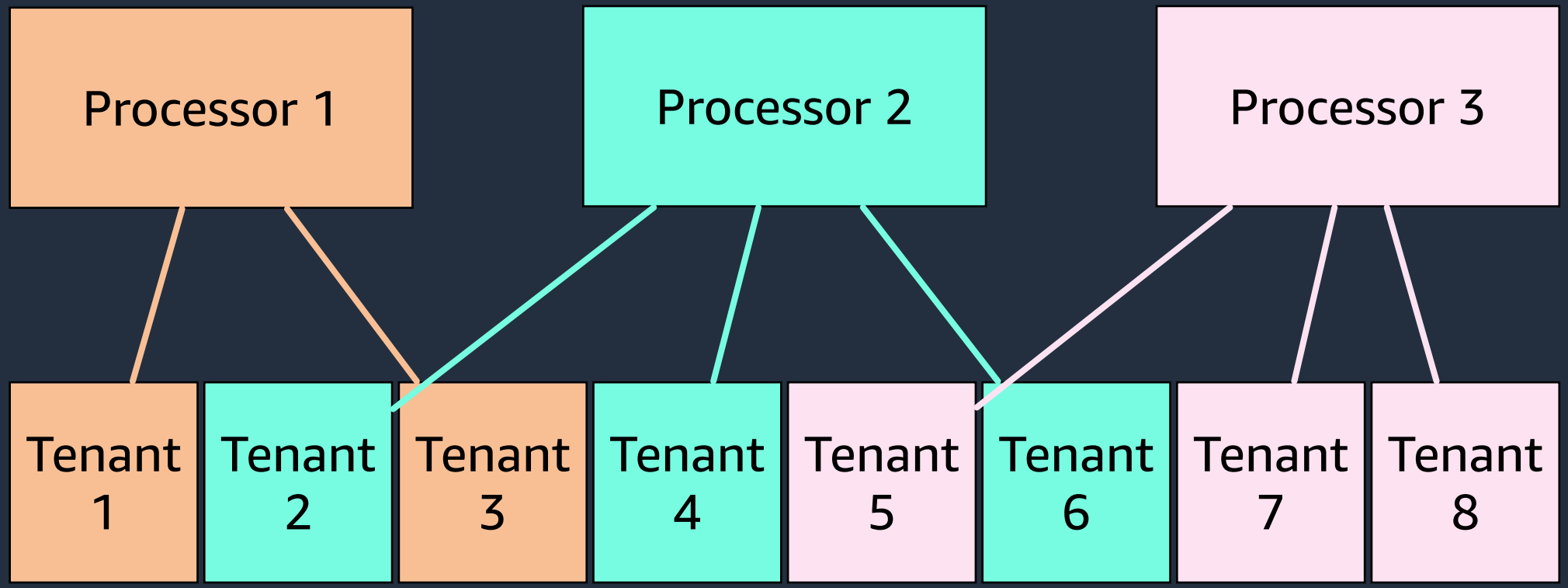
Let’s analyze the problem



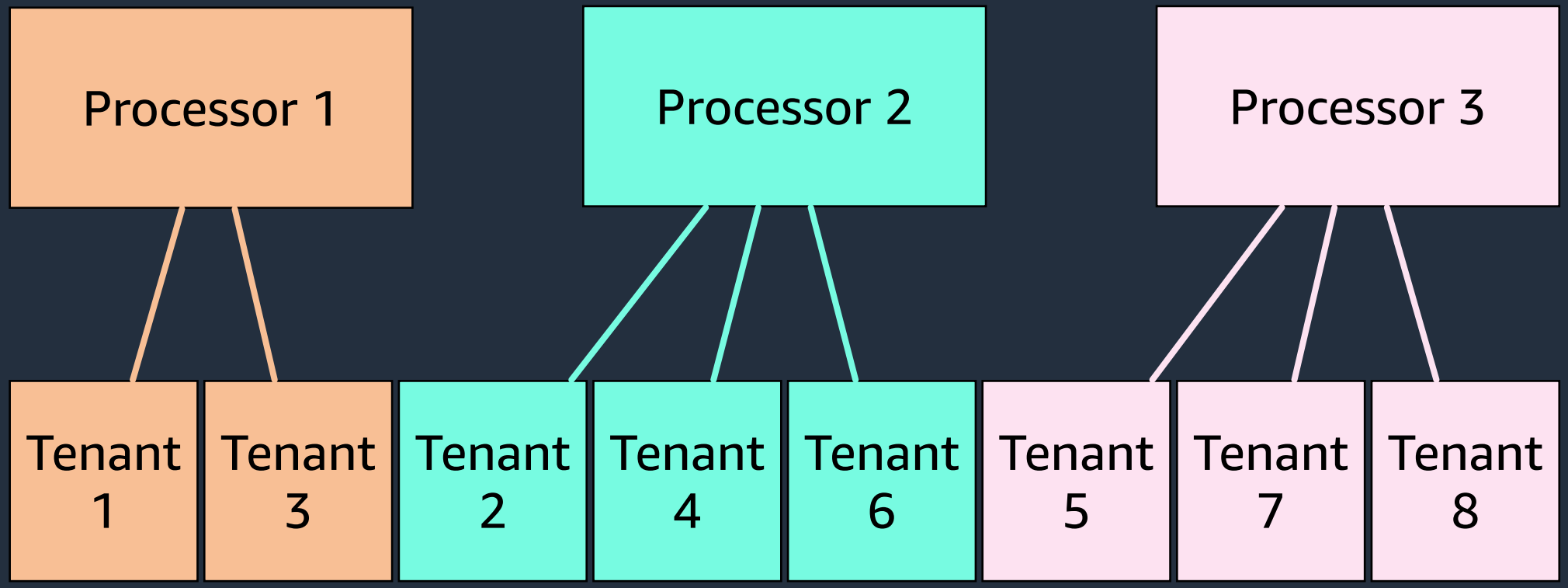
Let’s analyze the problem



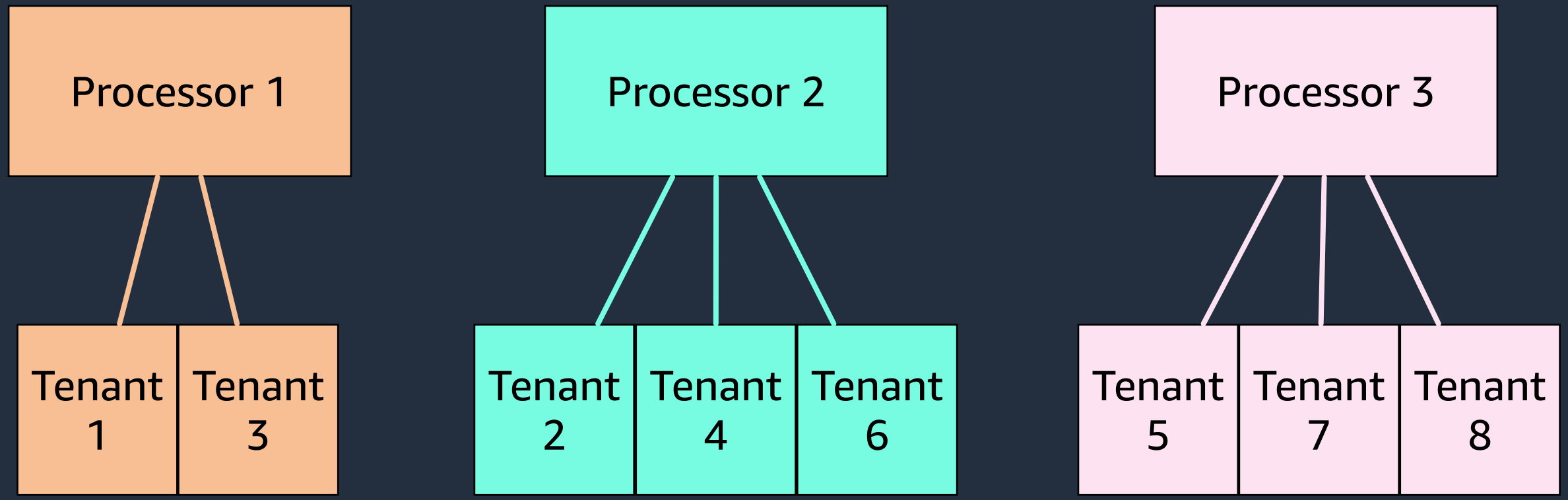
Let’s analyze the problem



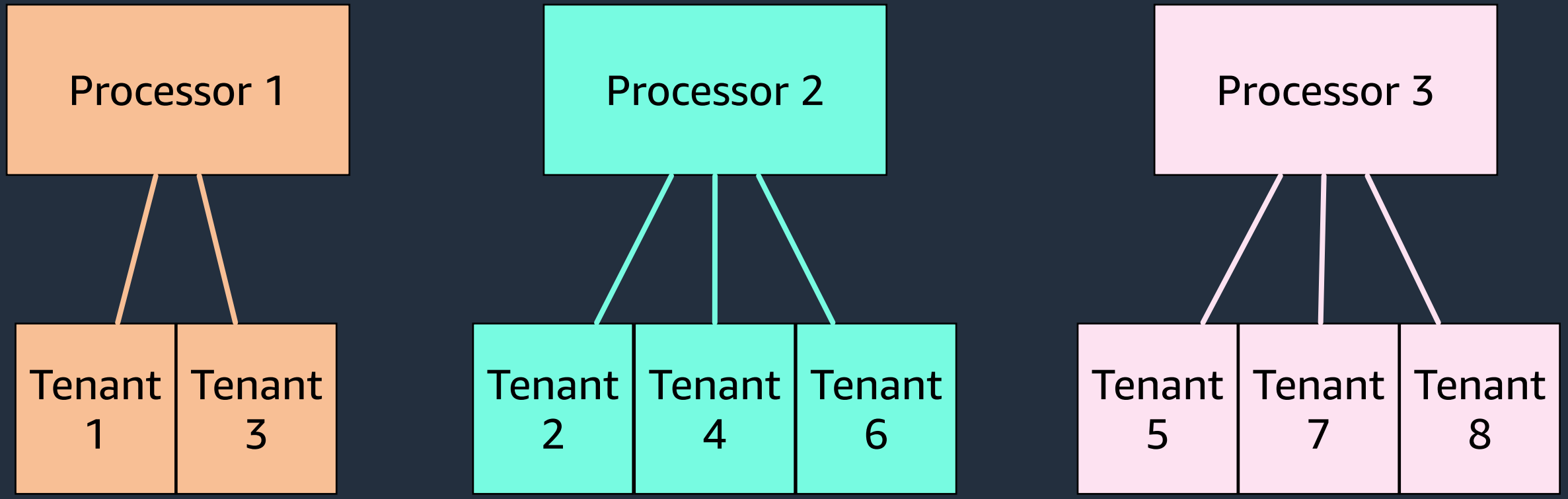
Let’s analyze the problem



Let’s analyze the problem

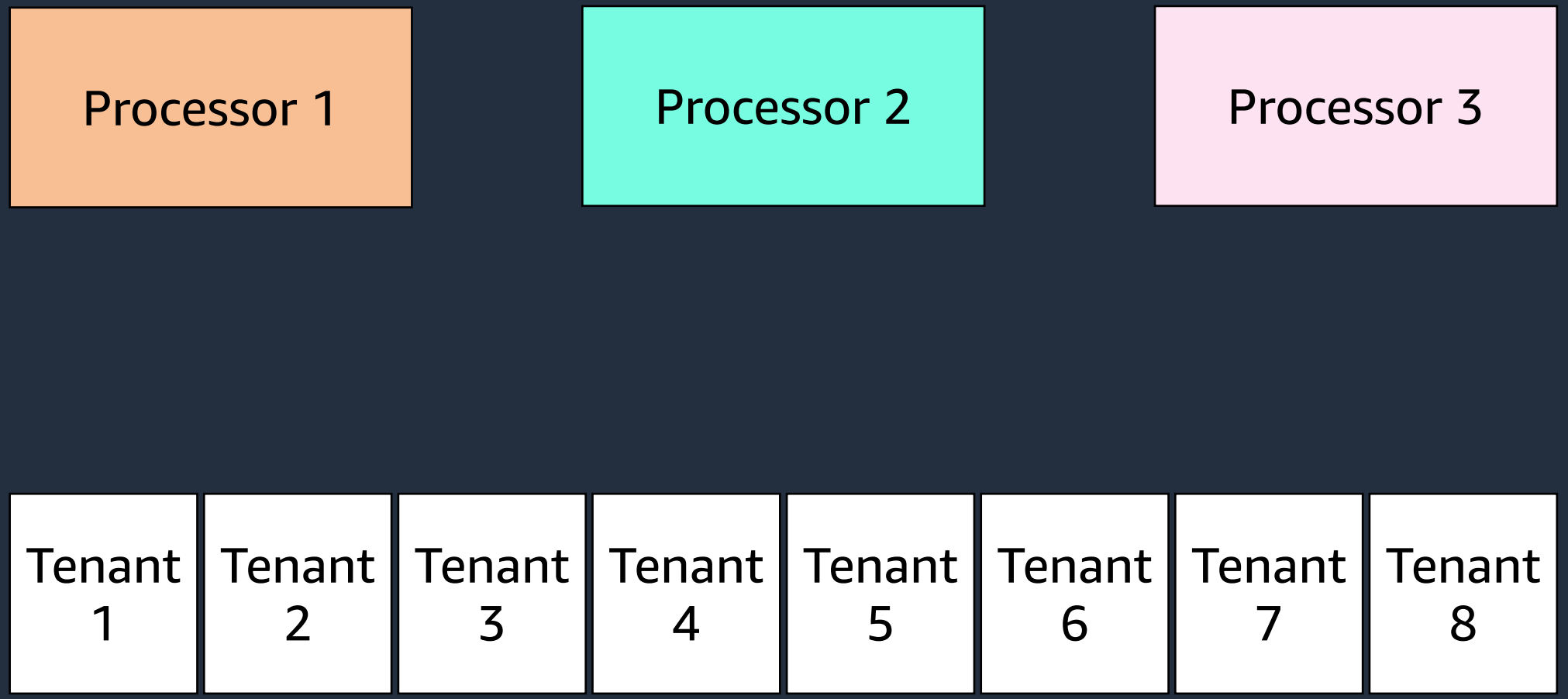


Let's analyze the problem

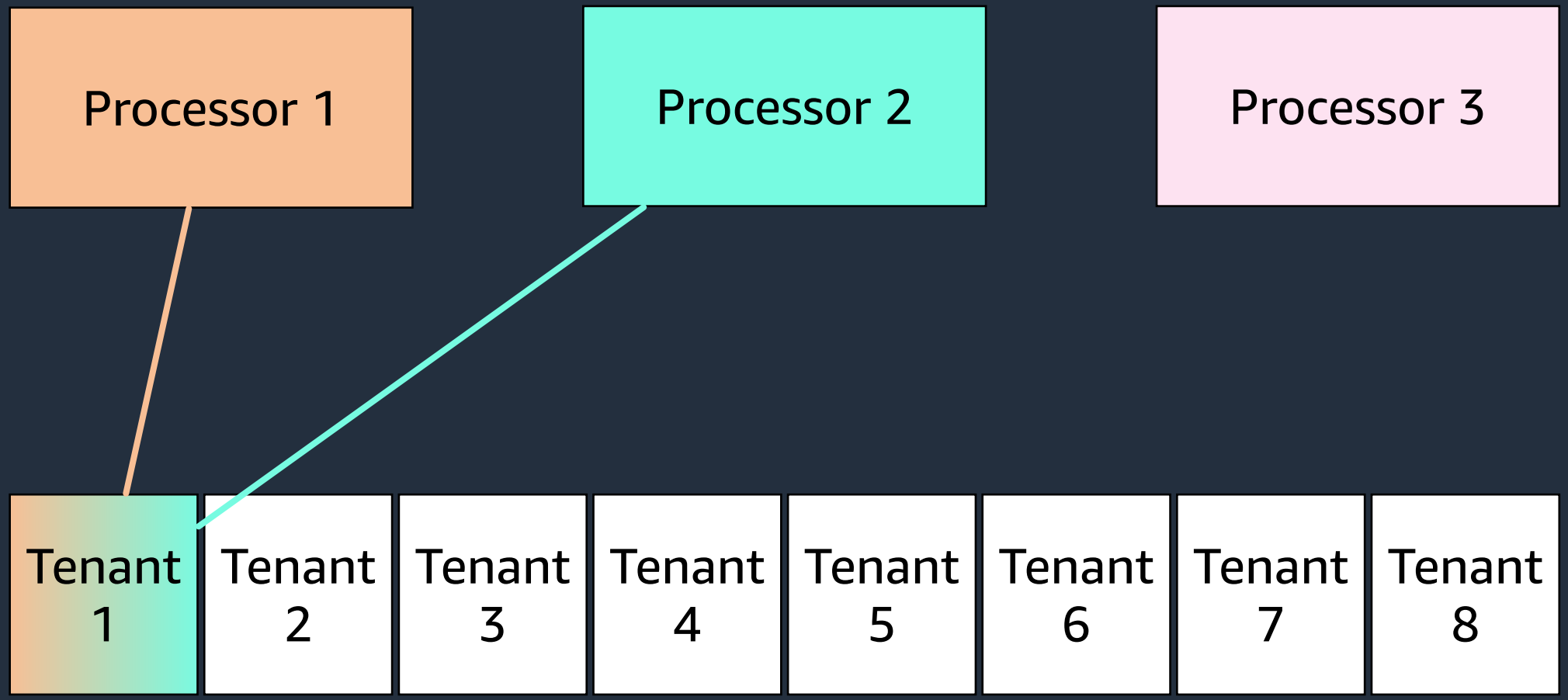


Same problem as before, but on a smaller scale

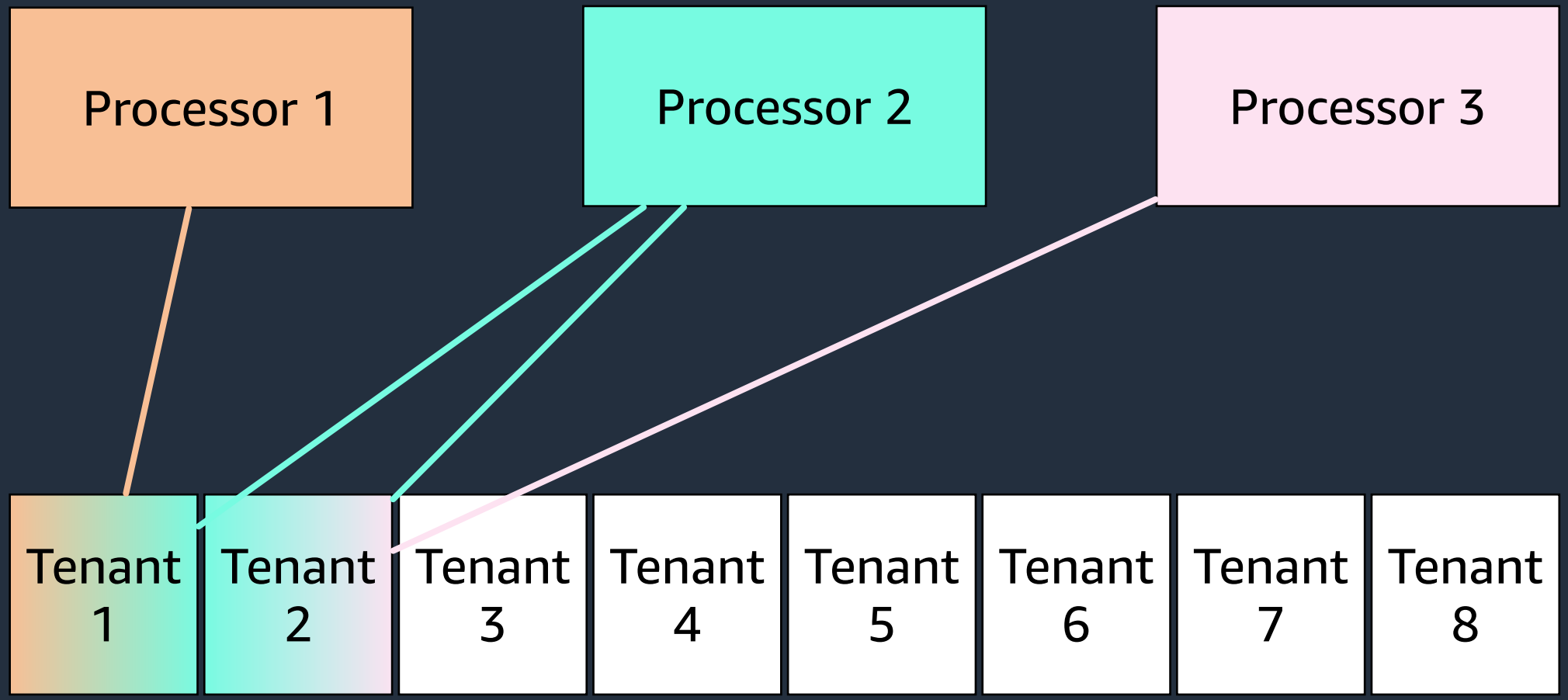
Shuffle sharding



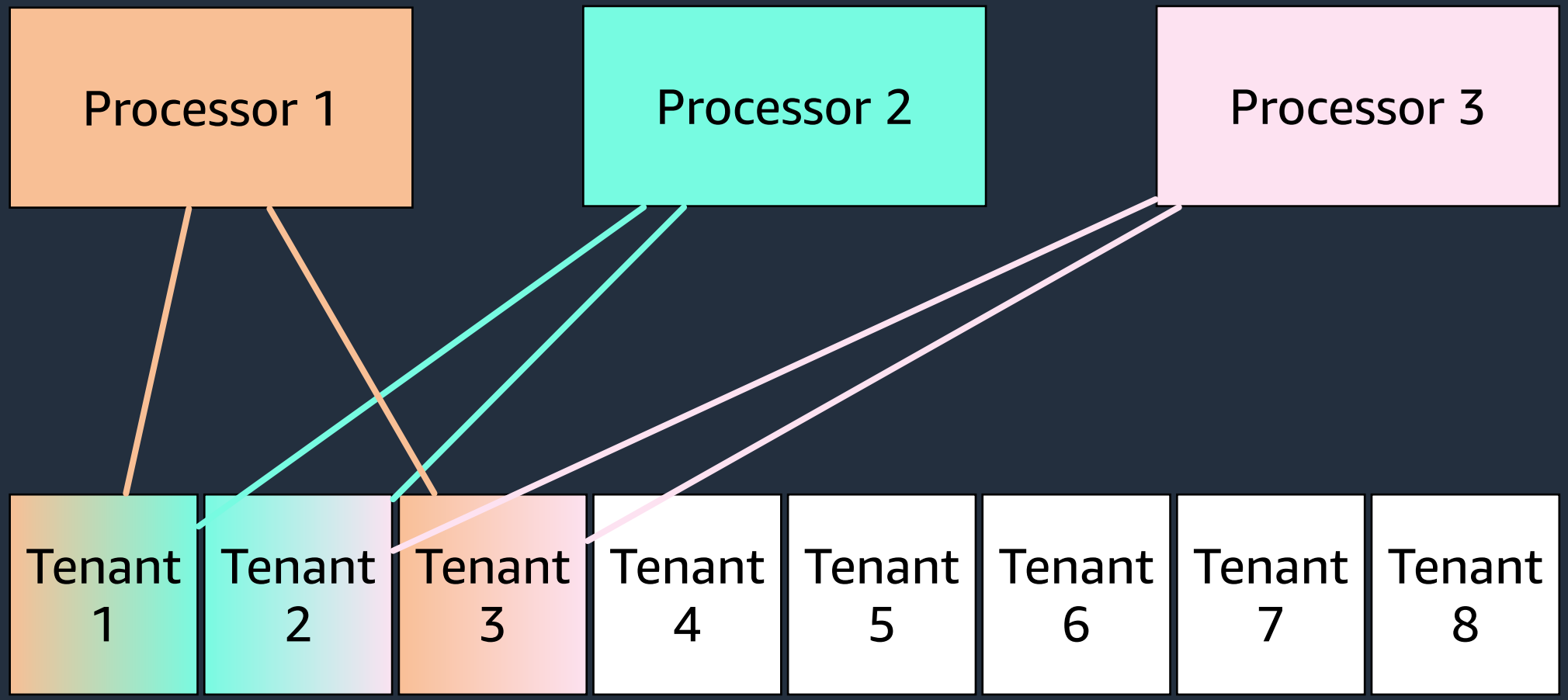
Shuffle sharding



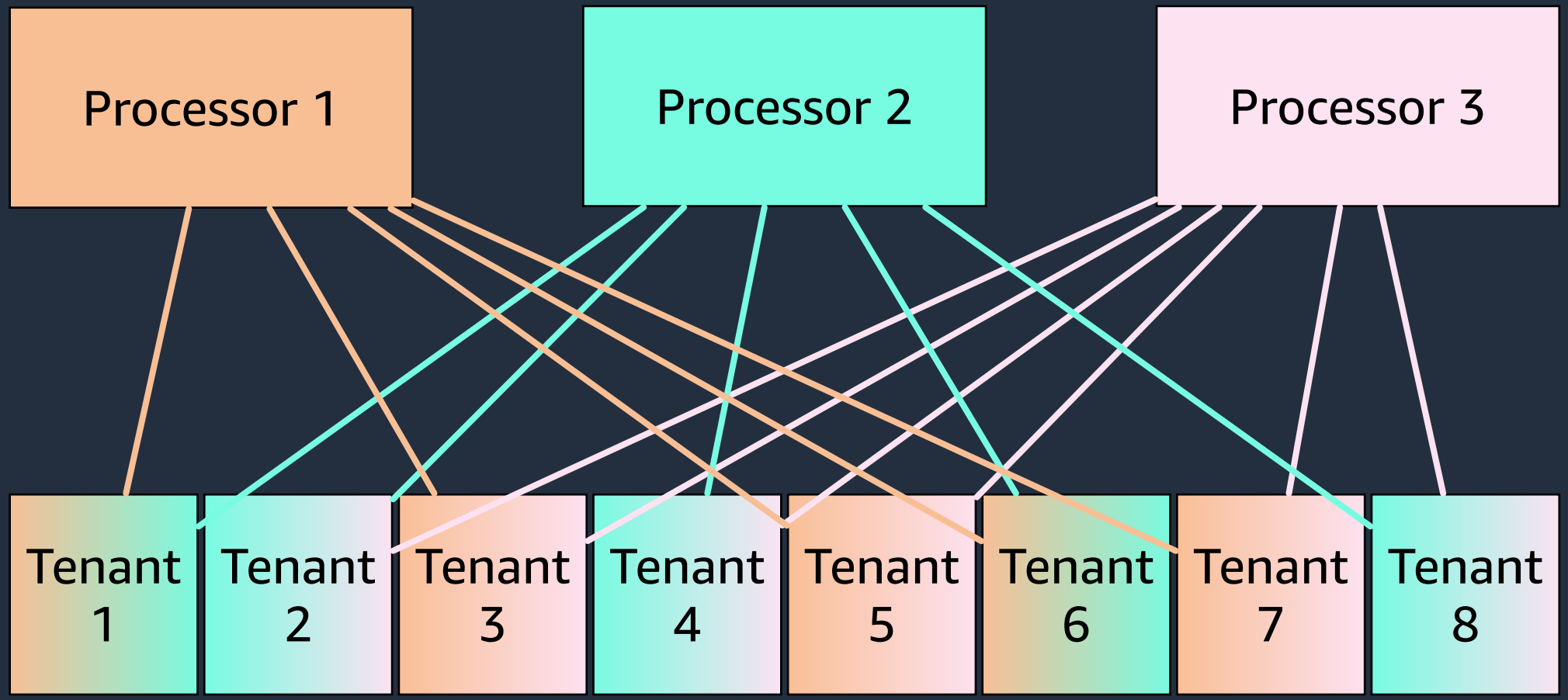
Shuffle sharding



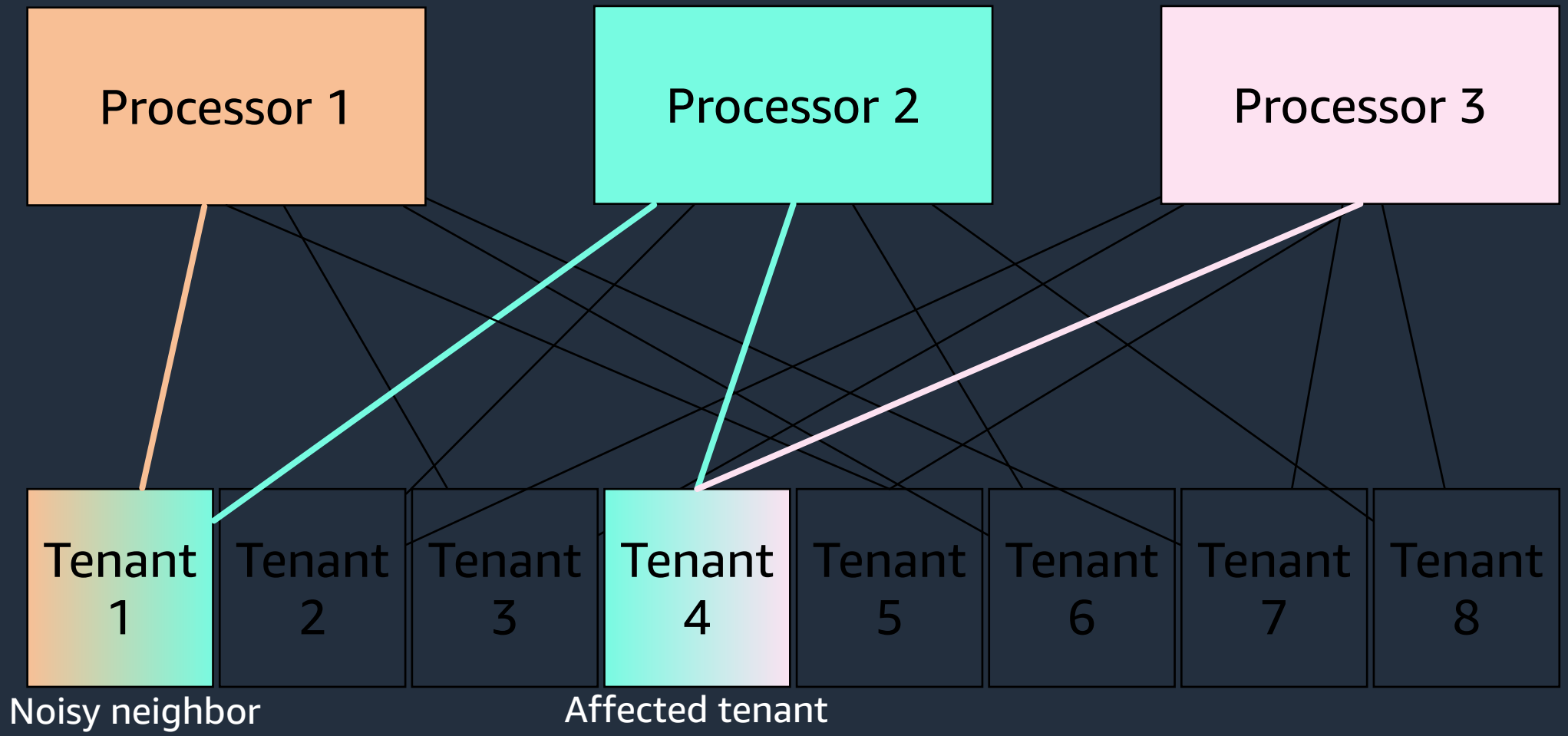
Shuffle sharding



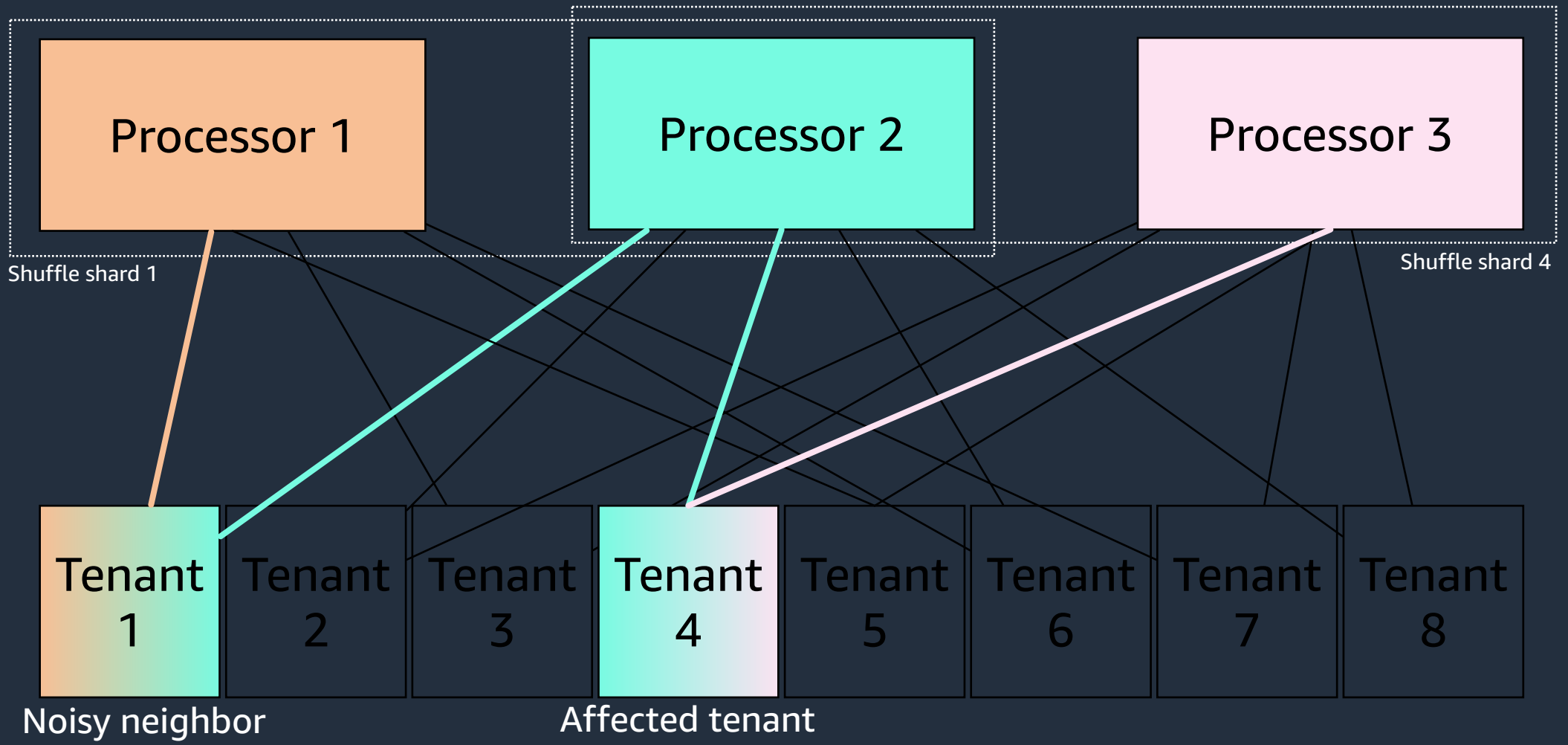
Shuffle sharding



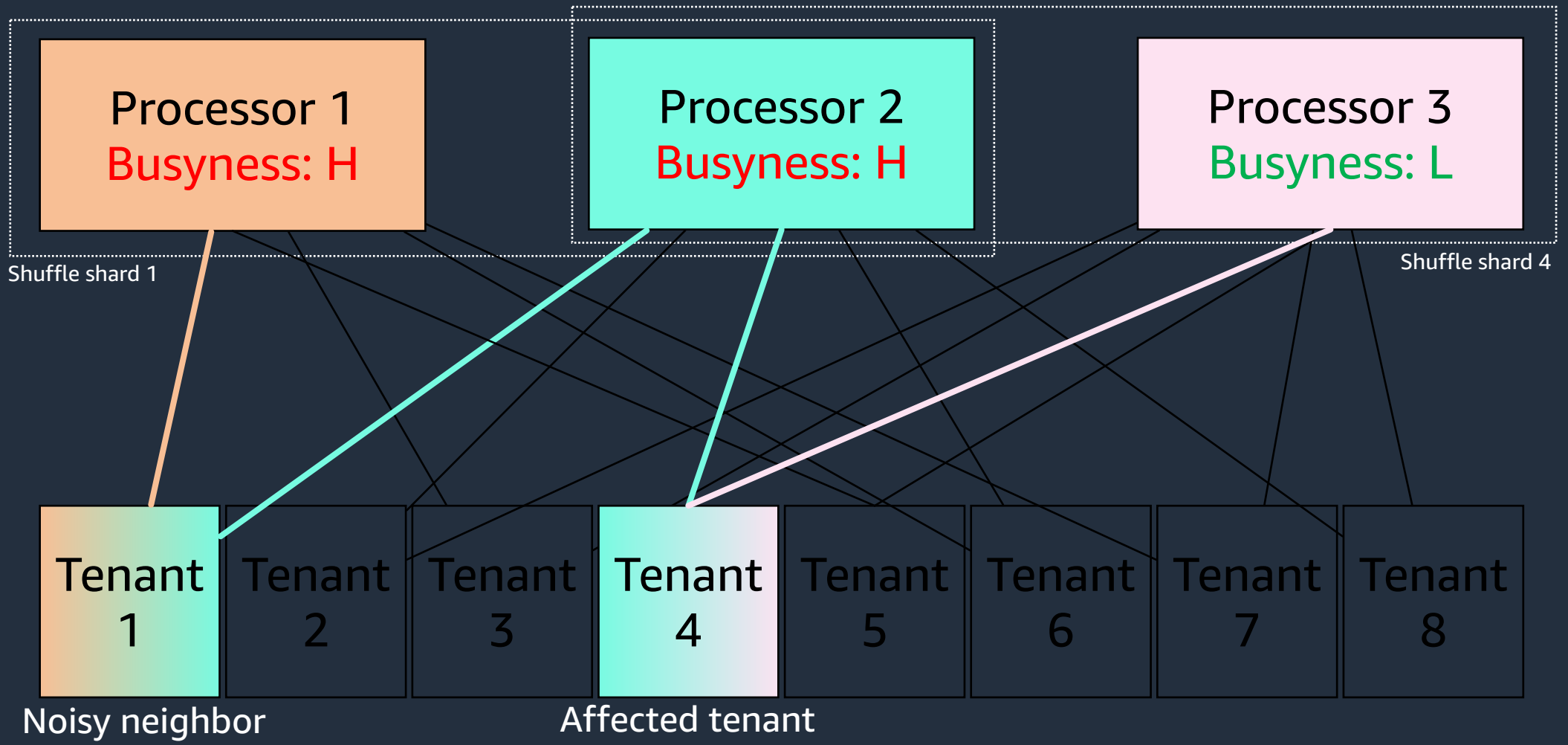
Shuffle sharding



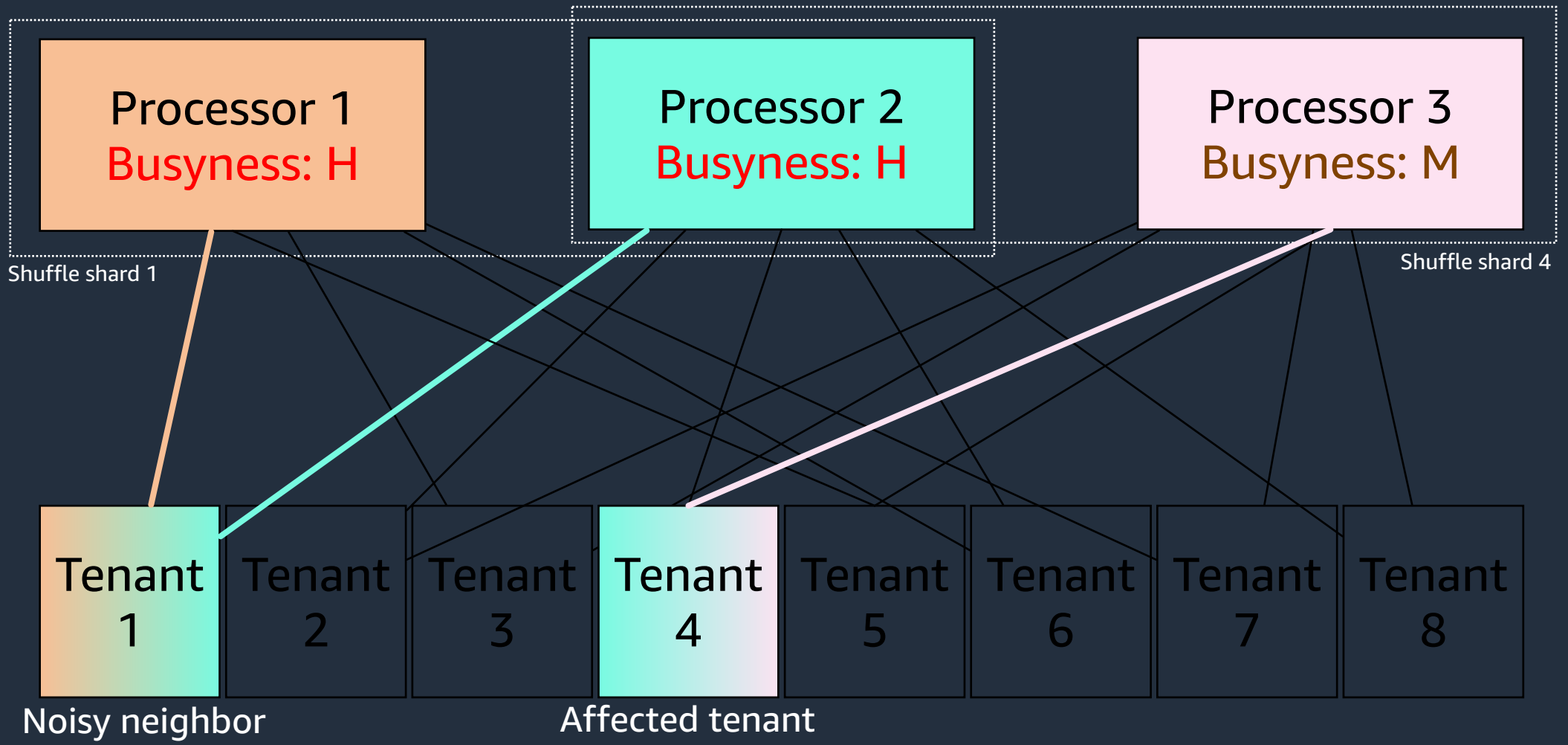
Shuffle sharding



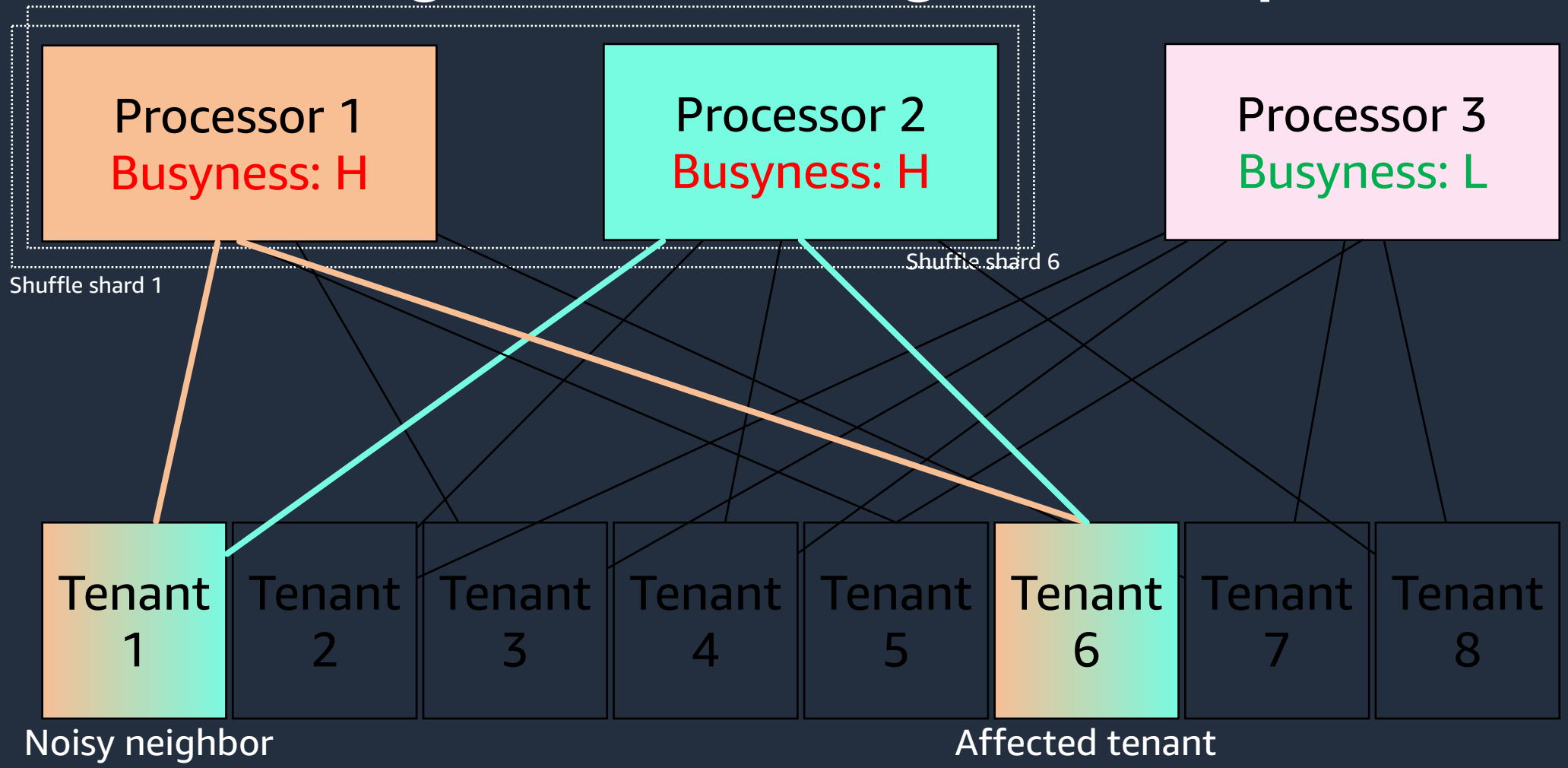
Shuffle sharding – adding a heuristic



Shuffle sharding – adding a heuristic



Shuffle sharding – tenants using the same processors



Probability of using the same processors set

p – number of processors
s – processors per tenant (shuffle shard size)
N = C(p, s) number of combinations
P_2 – probability of 2 tenants picking the same combination = $(1/N)^1$
P_3 – probability of 3 tenants picking the same combination = $(1/N)^2$
P_4 – probability of 4 tenants picking the same combination = $(1/N)^3$

(this is the most boring slide in the whole presentation)

Probability of using the same processors set

Number of processors	1	3	5	8	10	20	50
Processors per tenant (shard size)	1	1	2	3	3	3	3
Number of combinations	1	3	10	56	120	1140	19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Probability of using the same processors set

Number of processors	1	3	5	8	10	20	50
Processors per tenant (shard size)	1	1	2	3	3	3	3
Number of combinations	1	3	10	56	120	1140	19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Probability of using the same processors set

Number of processors	1	3	5	8	10	20	50
Processors per tenant (shard size)	1	1	2	3	3	3	3
Number of combinations	1	3	10	56	120	1140	19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Probability of using the same processors set

Number of processors	1	3	5	8	10	20	50
Processors per tenant (shard size)	1	1	2	3	3	3	3
Number of combinations	1	3	10	56	120	1140	19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Probability of using the same processors set

Number of processors	1	3	5	8	10	20	50
Processors per tenant (shard size)	1	1	2	3	3	3	3
Number of combinations	1	3	10	56	120	1140	19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Probability of using the same processors set

Number of processors	1	3					50
Processors per tenant (shard size)	1	1					3
Number of combinations	1	3					19600
P (2 tenants using same shard)	100%	33%	10%	2%	0.83%	0.0877%	0.00510%
P (3 tenants using same shard)	100%	11%	1%	0.0319%	0.0069%	0.0000769%	0.000000260%
P (4 tenants using same shard)	100%	4%	0.1%	0.0006%	0.00006%	0.000000067%	0.0000000000133%

Statistically

For ~20,000 tenants, a noisy neighbor should not have major impact on any other tenants

For ~100,000 tenants, a noisy neighbor, should only have major impact on 4 other tenants

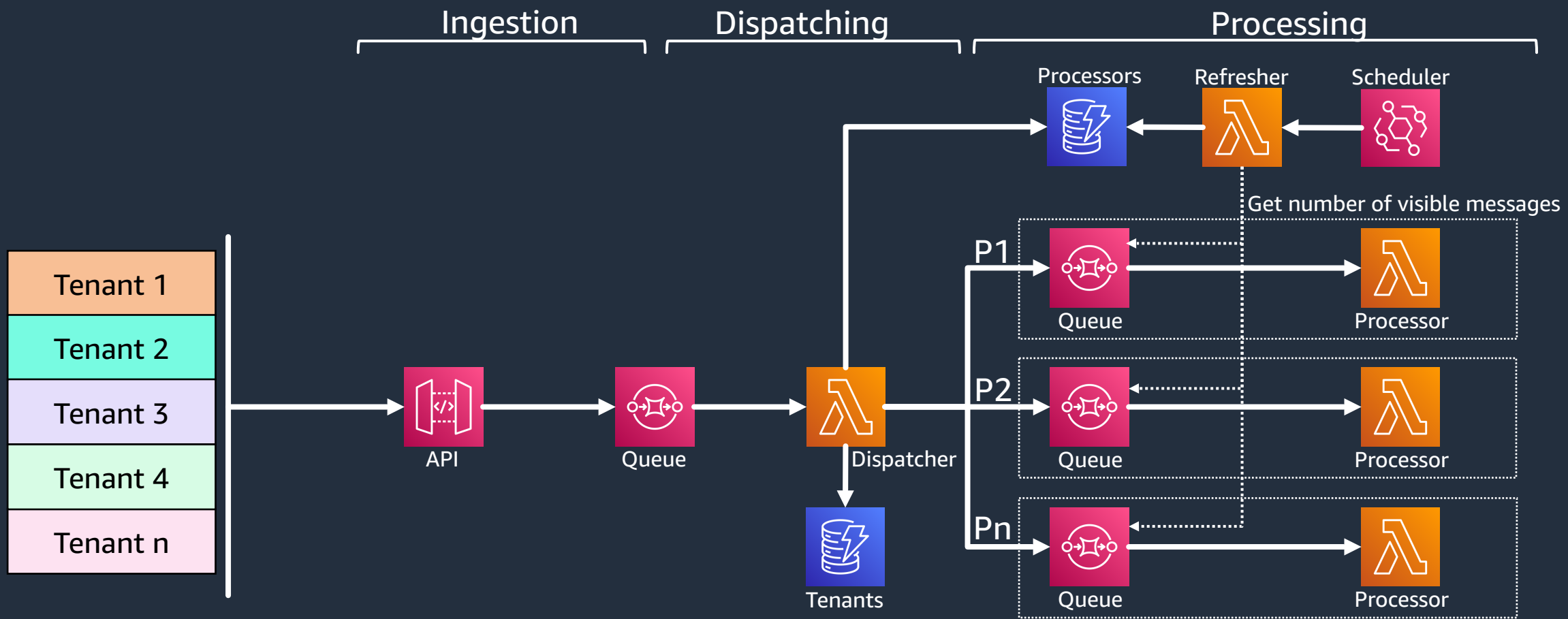
Shuffle sharding and Route 53

Amazon Route 53 and shuffle sharding

How does all of this help Amazon Route 53? With Route 53, we decided to arrange our capacity into a total of 2048 virtual name servers. These servers are virtual because they don't correspond to the physical servers hosting Route 53. We can move them around to help manage capacity. We then assign every customer domain to a shuffle shard of four virtual name servers. With those numbers, there are a staggering 730 billion possible shuffle shards. We have so many possible shuffle shards that we can assign a unique shuffle shard to every domain. Actually, we can go further, and ensure that no customer domain will ever share more than two virtual name servers with any other customer domain.

https://aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding/

Shuffle sharding



Let’s see this in action

Noisy neighbor!

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput
Request – Tenant 1	673	406	315	589	661	990	221	1282	0.00%	2.0/sec
Request – Tenant 10	653	402	311	580	665	864	225	1143	0.00%	1.9/sec
Request – Tenant 11	680	404	294	589	720	1084	228	1281	0.00%	2.0/sec
Request – Tenant 12	643	404	304	586	657	1032	232	1277	0.00%	1.9/sec
Request – Tenant 13	646	411	306	615	729	1043	230	1118	0.00%	1.9/sec
Request – Tenant 14	629	404	297	597	678	920	229	1213	0.00%	1.9/sec
Request – Tenant 15	664	393	291	578	654	865	222	1226	0.00%	2.0/sec
Request – Tenant 16	672	410	324	596	685	1011	227	1208	0.00%	2.0/sec
Request – Tenant 17	642	397	287	588	674	1051	228	1250	0.00%	1.9/sec
Request – Tenant 18	689	397	293	579	673	985	224	1141	0.00%	2.0/sec
Request – Tenant 19	655	405	293	590	711	1012	231	1150	0.00%	1.9/sec
Request – Tenant 20	9968	408	310	590	691	1060	220	1579	0.00%	29.3/sec
Request – Tenant 4	657	415	339	595	700	1051	231	1174	0.00%	2.0/sec
Request – Tenant 5	701	400	297	582	657	966	227	1198	0.00%	2.1/sec
Request – Tenant 6	723	403	294	596	673	1044	231	2183	0.00%	2.1/sec
Request – Tenant 7	669	408	304	590	719	1051	231	1294	0.00%	2.0/sec
Request – Tenant 8	692	401	305	588	671	920	233	1168	0.00%	2.0/sec
Request – Tenant 9	710	407	289	597	689	1045	225	1661	0.00%	2.1/sec
TOTAL	22690	405	304	590	690	1039	220	2183	0.00%	66.7/sec

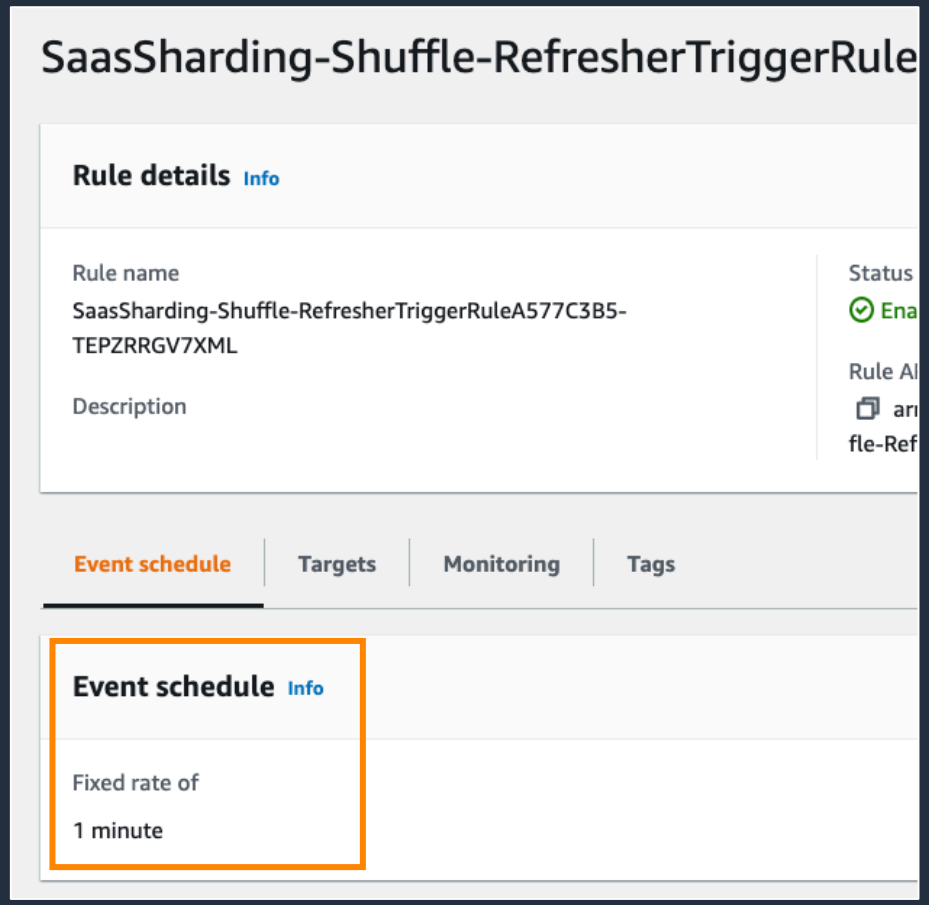
Before we begin

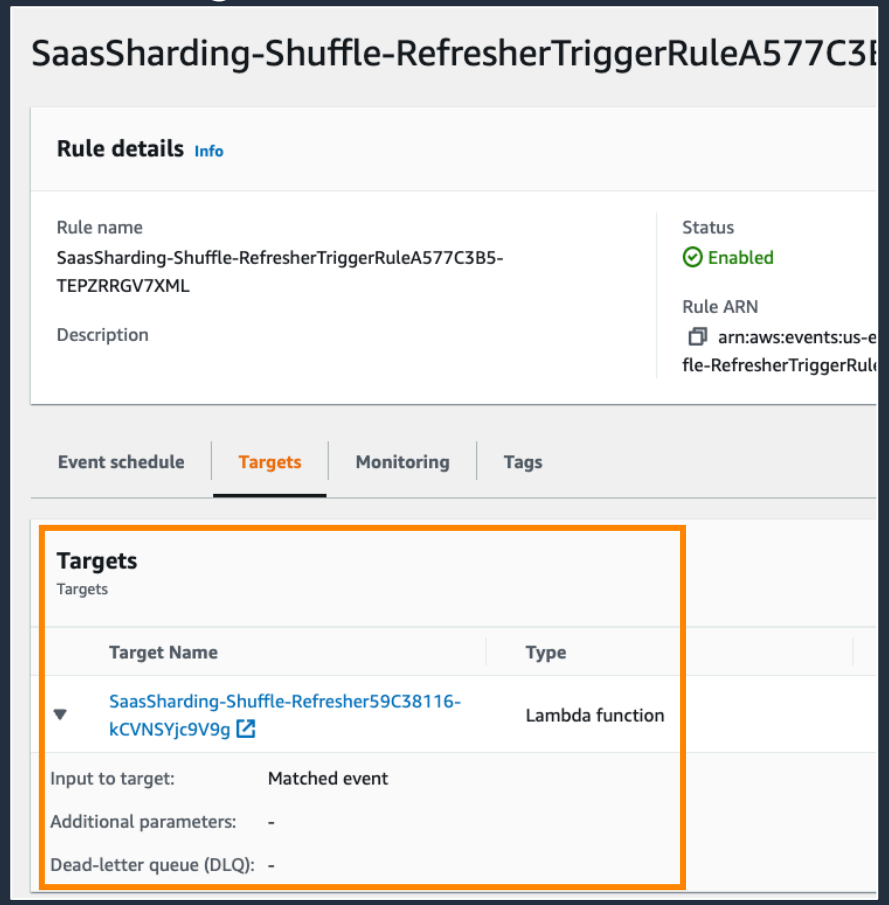
Processor Queues - SQS

processorId	approximateNumberOfMessagesVisible	queueUrl
1	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P1DataProcessorQueue1D1C95DF-LKCok95fqW5V
2	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P2DataProcessorQueue880A1E3E-ViaI6DMprNSp
3	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P3DataProcessorQueue2DACC5EB-6WE276nFSVqd
4	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P4DataProcessorQueue8B458582-roCPPSu23Z2Q
5	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P5DataProcessorQueue799FB189-vHvHj55E5zHO
6	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P6DataProcessorQueueC6D8E8E5-AjcPjl4ZHtQk
7	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P7DataProcessorQueueF298AEC3-e9tZJ8zo93sA
8	0	https://sqs.us-east-1.amazonaws.com/281024298475/SaasSharding-Shuffle-P8DataProcessorQueue1F68FCB8-Xd7m6712kW1d

Before we begin

Refresher trigger - EventBridge





Before we begin

Refresher function - Lambda

```
function findBestProcessor(tenantId){
  console.log('> findBestProcessor for tenant', tenantId);
  const tenantProcessorIds = tenants[tenantId];

  // Get processors for this tenantId
  let tenantProcessors = tenantProcessorIds.map(id=>processors[id]);

  // Sort processors by number of messages visible
  tenantProcessors.sort((a,b)=>a.approximateNumberOfMessagesVisible - b.approximateNumberOfMessagesVisible);

  // Get the smallest number of messages visible number
  const min = tenantProcessors[0].approximateNumberOfMessagesVisible;

  // Filter out all processors with number of messages !== min
  const filteredProcessors = tenantProcessors.filter((i)=>i.approximateNumberOfMessagesVisible===min);

  // Pick random processor from the remaining list
  const randomIdx = Math.floor(Math.random() * filteredProcessors.length);
  const bestProcessor = filteredProcessors[randomIdx];

  return bestProcessor;

}
```

Before we begin

tenantId	processorIds
1	[2,1,6]
10	[2,4,7]
11	[3,2,1]
12	[1,5,8]
13	[1,4,5]
14	[7,8,2]
15	[1,8,6]
16	[2,6,7]
17	[4,5,2]
18	[1,2,5]
19	[4,5,3]
2	[4,8,3]
20	[7,8,6]
3	[6,8,5]
4	[1,3,6]
5	[3,2,8]
6	[2,7,4]
7	[5,1,4]
8	[3,5,6]
9	[4,3,7]

Tenants Table - DynamoDB

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
P1	X			X			X				X	X	X		X			X		
P2	X				X	X				X	X			X		X	X	X		
P3		X		X	X			X	X		X								X	
P4		X				X	X		X	X			X				X		X	
P5			X				X	X				X	X				X	X	X	
P6	X		X	X				X							X	X				X
P7						X			X	X				X		X				X
P8		X	X		X							X		X	X					X

Before we begin

tenantId	processorIds
1	[2,1,6]
10	[2,4,7]
11	[3,2,1]
12	[1,5,8]
13	[1,4,5]
14	[7,8,2]
15	[1,8,6]
16	[2,6,7]
17	[4,5,2]
18	[1,2,5]
19	[4,5,3]
2	[4,8,3]
20	[7,8,6]
3	[6,8,5]
4	[1,3,6]
5	[3,2,8]
6	[2,7,4]
7	[5,1,4]
8	[3,5,6]
9	[4,3,7]

Tenants Table - DynamoDB

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
P1	X			X			X				X	X	X		X			X		
P2	X				X	X				X	X			X		X	X	X		
P3		X		X	X			X	X		X								X	
P4		X				X	X		X	X			X				X		X	
P5			X				X	X				X	X				X	X	X	
P6	X		X	X				X							X	X				X
P7						X			X	X				X		X				X
P8		X	X		X							X		X	X					X

Before we begin

tenantId	processorIds
1	[2,1,6]
10	[2,4,7]
11	[3,2,1]
12	[1,5,8]
13	[1,4,5]
14	[7,8,2]
15	[1,8,6]
16	[2,6,7]
17	[4,5,2]
18	[1,2,5]
19	[4,5,3]
2	[4,8,3]
20	[7,8,6]
3	[6,8,5]
4	[1,3,6]
5	[3,2,8]
6	[2,7,4]
7	[5,1,4]
8	[3,5,6]
9	[4,3,7]

Tenants Table - DynamoDB

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
P1	X			X			X				X	X	X		X			X		
P2	X				X	X				X	X			X		X	X	X		
P3		X		X	X			X	X		X								X	
P4		X				X	X		X	X			X				X		X	
P5			X				X	X				X	X				X	X	X	
P6	X		X	X				X							X	X				X
P7						X			X	X				X		X				X
P8		X	X		X							X		X	X					X

- 6 tenants not affected
- 13 tenants affected

Before we begin

Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, T3, T4, T5, T6, T8, T9, T10, T11, T14, T15, T16

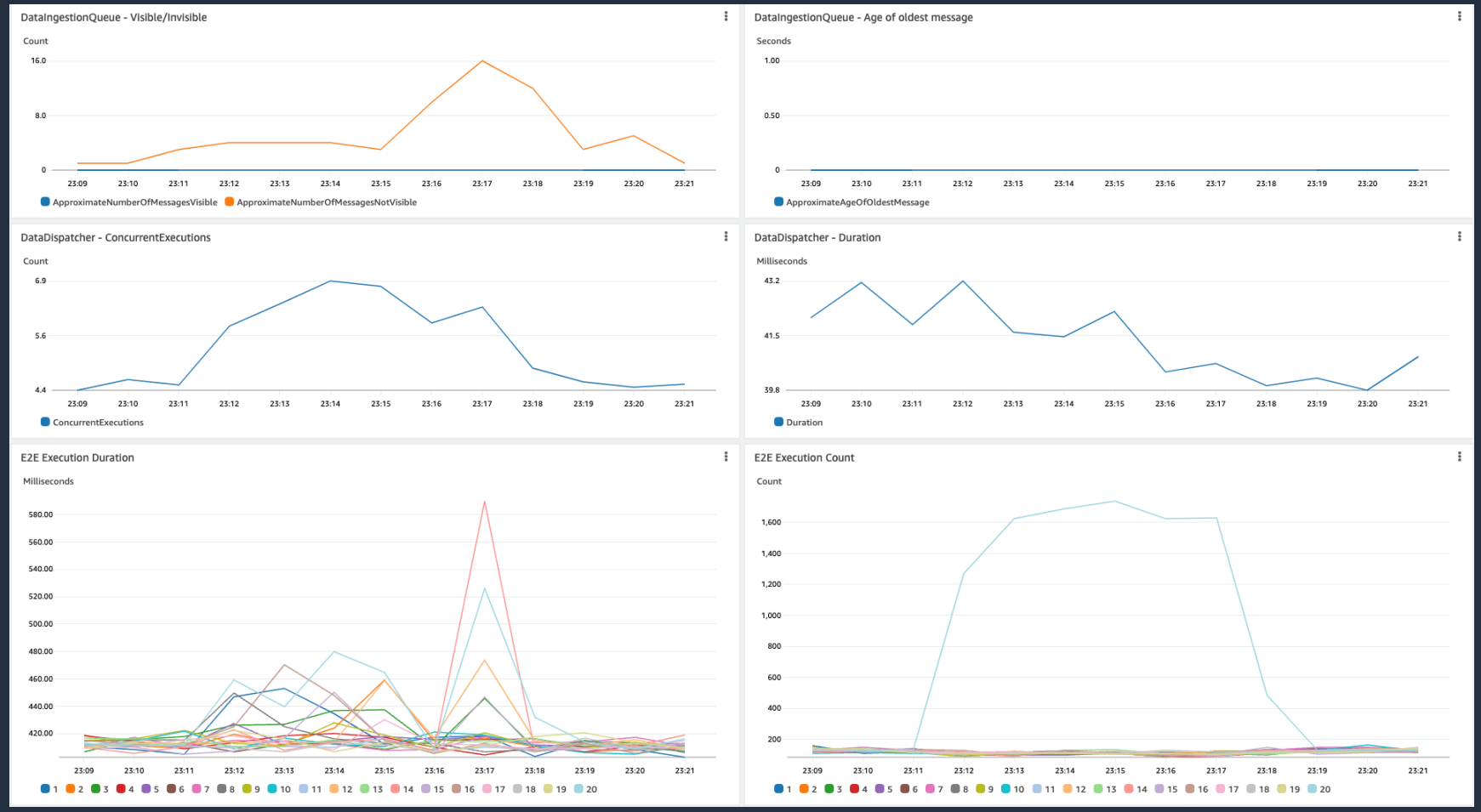
tenantId	processorIds
1	[2,1,6]
10	[2,4,7]
11	[3,2,1]
12	[1,5,8]
13	[1,4,5]
14	[7,8,2]
15	[1,8,6]
16	[2,6,7]
17	[4,5,2]
18	[1,2,5]
19	[4,5,3]
2	[4,8,3]
20	[7,8,6]
3	[6,8,5]
4	[1,3,6]
5	[3,2,8]
6	[2,7,4]
7	[5,1,4]
8	[3,5,6]
9	[4,3,7]

Tenants Table - DynamoDB

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T19	T20
P1	X			X			X				X	X	X		X		X	
P2	X				X	X				X	X			X		X	X	X	
P3		X		X	X			X	X		X							X	
P4		X				X	X		X	X			X				X	X	
P5			X				X	X				X	X				X	X	X	
P6	X		X	X				X							X	X				X
P7						X			X	X				X		X				X
P8		X	X		X							X		X	X					X

- 6 tenants not affected
- 13 tenants affected

Results



Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, T3, T4, T5, T6, T8, T9, T10, T11, T14, T15, T16

Results

Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, T3, T4, T5, T6, T8, T9, T10, T11, T14, T15, T16

Peace and quiet

Noisy neighbor

Cooldown

Results

Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, **T3**, T4, T5, T6, T8, T9, T10, T11, **T14**, **T15**, **T16**

Peace and quiet

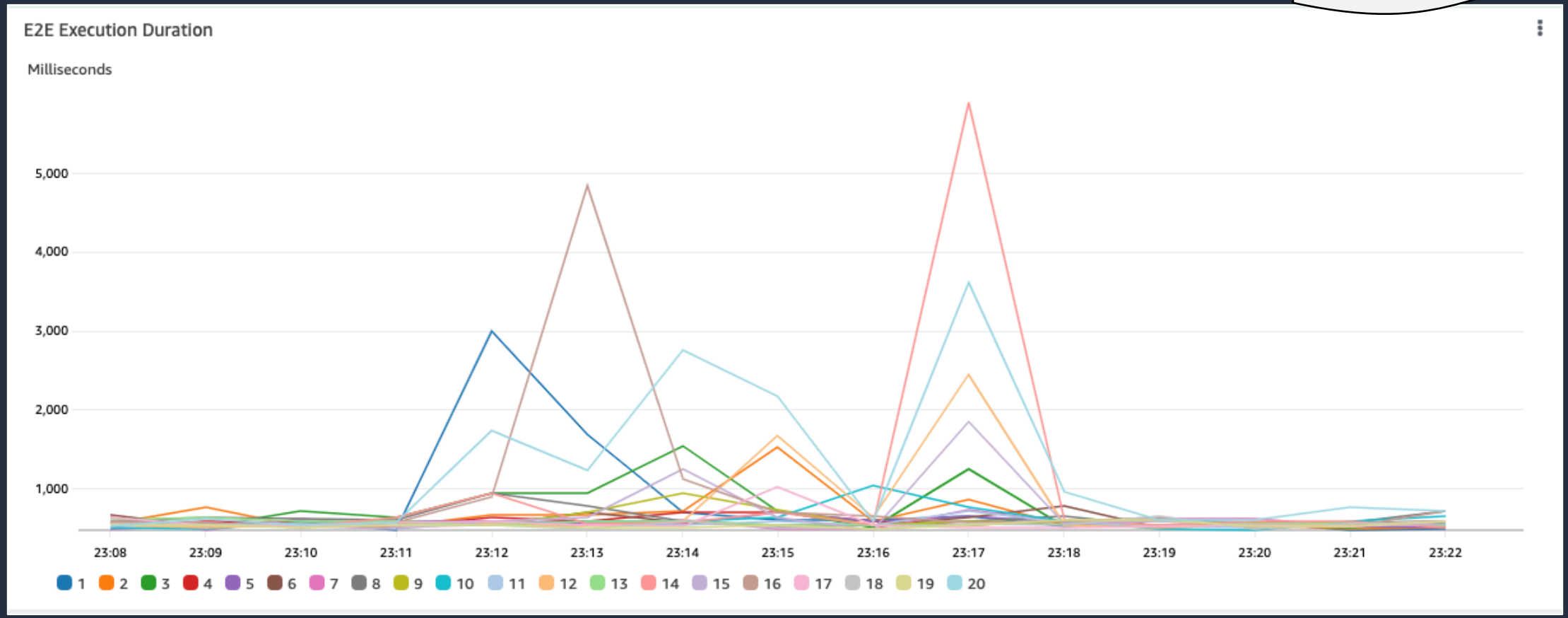
Noisy neighbor

Cooldown

Results

Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, T3, T4, T5, T6, T8, T9, T10, T11, T14, T15, T16

P99



Results

Mental note
Noisy neighbor – 20
Affected processors – P6, P7, P8
Affected tenants – T1, T2, T3, T4, T5, T6, T8, T9, T10, T11, T14, T15, T16

Conclusions

- Shuffle sharding helps by adding more resiliency to the system
- Built for containing blast radius and isolating failures
- Random tenant-to-shard assignment limits customer impact
- Using heuristics for rebalancing can help to build a self-healing system
- Do more with less

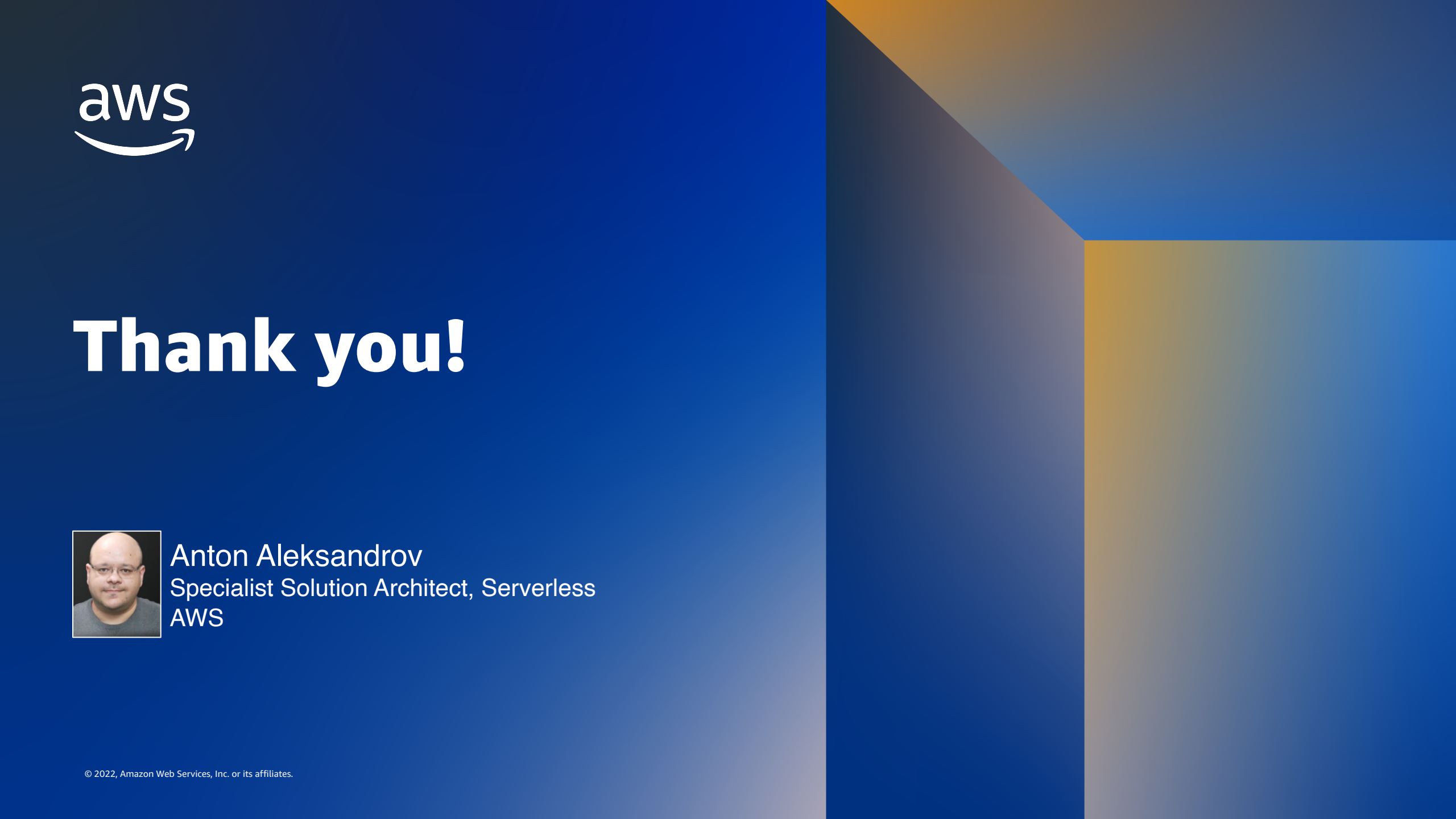
Summary

Action items

- Sharding is a great technique for fault isolation and achieving higher workflow resiliency
- Shuffle sharding helps to limit the blast radius exponentially
- Hope for success, build for failure

Additional Shuffle Sharding content

- Workload isolation using shuffle-sharding https://tinyurl.com/shufflesharding1
- Shuffle Sharding: Massive and Magical Fault Isolation https://tinyurl.com/shufflesharding2
- Implement shuffle sharding lab https://tinyurl.com/shufflesharding3
- Improving Workload Resiliency Using Shuffle-Sharding https://youtu.be/TVdxPF7KL1c
- How to scale beyond limits with cell-based architectures https://youtu.be/HUwz8uko7HY



Thank you!

Anton Aleksandrov
Specialist Solution Architect, Serverless
AWS