Containment Domains within SWARM

Aaron Myles Landwehr
In collaboration with:
Sam Kaplan
Sergio Pino
Guang R. Gao

Outline

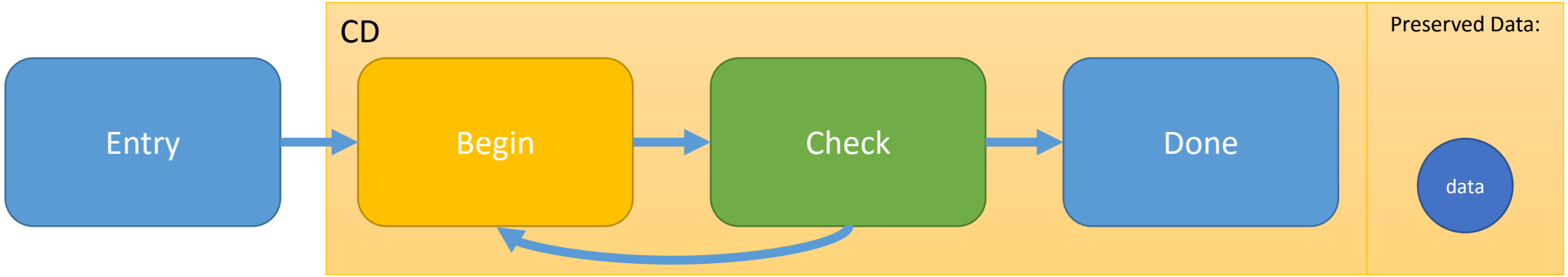
- Background on containment domains
- Semantics of containment domains
  - APIs
  - Preservation
- Example usage of APIs
- Nested CD Example
- Conclusion and Publications

Background and Motivation

- Containment Domains (CDs) address the weaknesses of checkpoint and restart
  - CDs are non-interval dependent unlike generic checkpointing which has little programmer control and can be inflexible to application needs
  - Allow the programmer and software system to tune the location and method preservation and recovery at the desired level of reliability while also maximizing performance of the system
  - Not susceptible to domino effect (full rollback) due to transactional characteristics

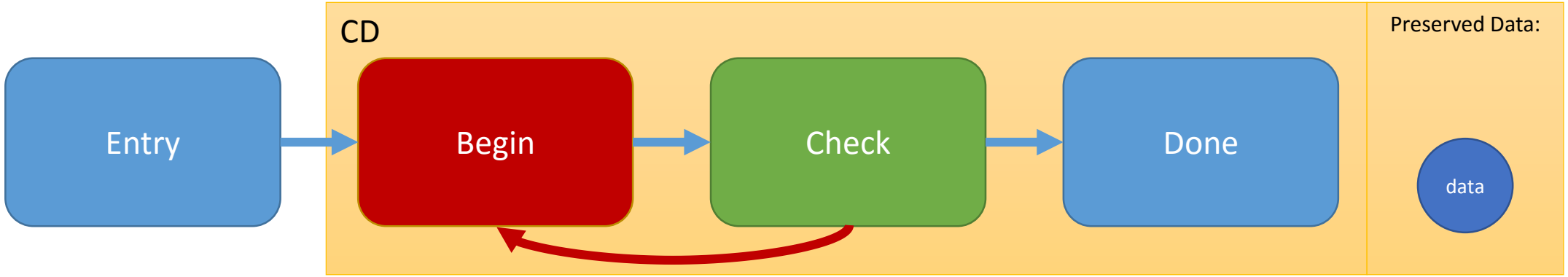
Containment Domain Semantics

- Conceptually there are three parts to a CD:
  - A ‘begin’ codelet to do actual problem execution.
  - A ‘check’ codelet to verify the results of the execution.
  - A ‘done’ codelet to run if the check succeeds.



Containment Domain Semantics

- On failure, the CD will be re-executed from Beginning.
  - Note, even in a more complex graph consisting of multiple codelets before the check codelet, the CD would re-run from the beginning.



Containment Domain APIs

- **void** swarm_ContainmentDomain_init(**swarm_ContainmentDomain_t * const** THIS)
- **void** swarm_ContainmentDomain_begin(**swarm_ContainmentDomain_t * const** THIS,
  const struct swarm_Codelet * begin, **void * const** begin_context,
  const struct swarm_Codelet * check, **void * const** check_context,
  const struct swarm_Codelet * done, **void * const** done_context)
- **void** swarm_ContainmentDomain_preserve(**swarm_ContainmentDomain_t * const** THIS,
  void * const data,
  const size_t length,
  swarm_int64 tid,
  const swarm_ContainmentDomain_preservation_type_t type)

Containment Domain APIs

- **void** swarm_ContainmentDomain_init(**swarm_ContainmentDomain_t * const** THIS)
- **void** swarm_ContainmentDomain_begin(**swarm_ContainmentDomain_t * const** THIS,
  const struct swarm_Codelet * begin, **void * const** begin_context,
  const struct swarm_Codelet * check, **void * const** check_context,
  const struct swarm_Codelet * done, **void * const** done_context)
- **void** swarm_ContainmentDomain_preserve(**swarm_ContainmentDomain_t * const** THIS,
  void * const data,
  const size_t length,
  swarm_int64 tid,
  const swarm_ContainmentDomain_preservation_type_t type)

- Initialize a containment domain

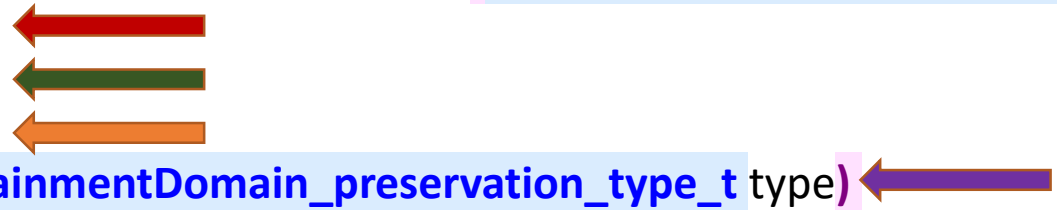
Containment Domain APIs

- **void** swarm_ContainmentDomain_init(**swarm_ContainmentDomain_t * const** THIS)
- **void** swarm_ContainmentDomain_begin(**swarm_ContainmentDomain_t * const** THIS,
  const struct swarm_Codelet * begin, **void * const** begin_context,
  const struct swarm_Codelet * check, **void * const** check_context,
  const struct swarm_Codelet * done, **void * const** done_context)
- **void** swarm_ContainmentDomain_preserve(**swarm_ContainmentDomain_t * const** THIS,
  void * const data,
  const size_t length,
  swarm_int64 tid,
  const swarm_ContainmentDomain_preservation_type_t type)

- Activate the specified domain
- Specify the 'begin' codelet
- Specify the 'check' codelet
- Specify the 'done' codelet

Containment Domain APIs

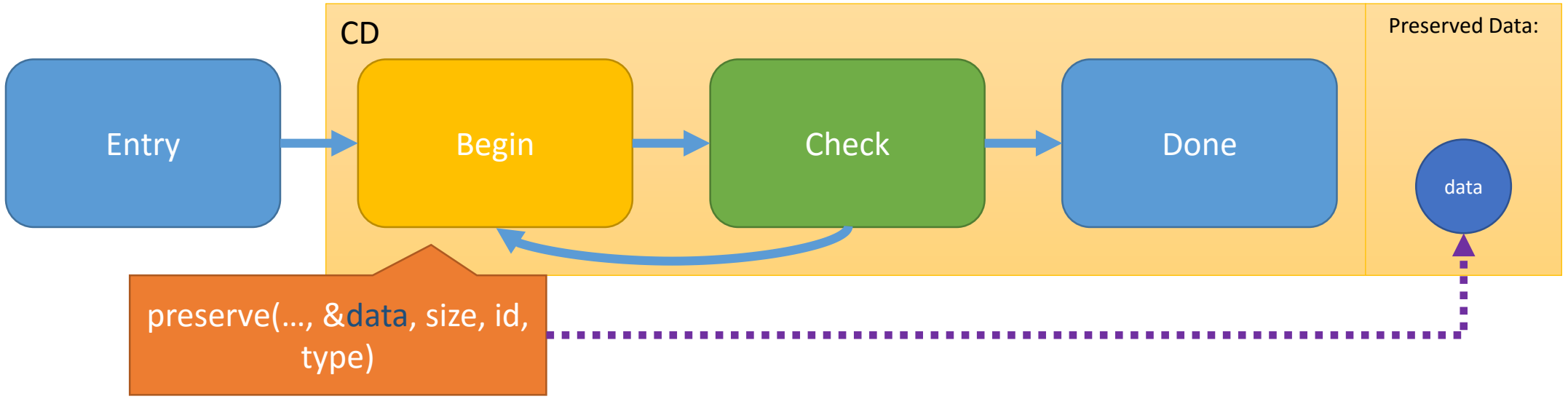
- `void swarm_ContainmentDomain_init(swarm_ContainmentDomain_t * const THIS)`
- `void swarm_ContainmentDomain_begin(swarm_ContainmentDomain_t * const THIS,`
  `const struct swarm_Codelet * begin, void * const begin_context,`
  `const struct swarm_Codelet * check, void * const check_context,`
  `const struct swarm_Codelet * done, void * const done_context)`
- `void swarm_ContainmentDomain_preserve(swarm_ContainmentDomain_t * const THIS,`
  `void * const data,`
  `const size_t length,`
  `swarm_int64 tid,`
  `const swarm_ContainmentDomain_preservation_type_t type)`



- Specify the data to perserve
- Specify the length of the data to perserve
- Specify an index associated with the data; this must be unique for all data preserved within a CD
- Specify the type of preservation; Either swarm_CONTAINMENTDOMAIN_COPY or swarm_CONTAINMENTDOMAIN_PARENT

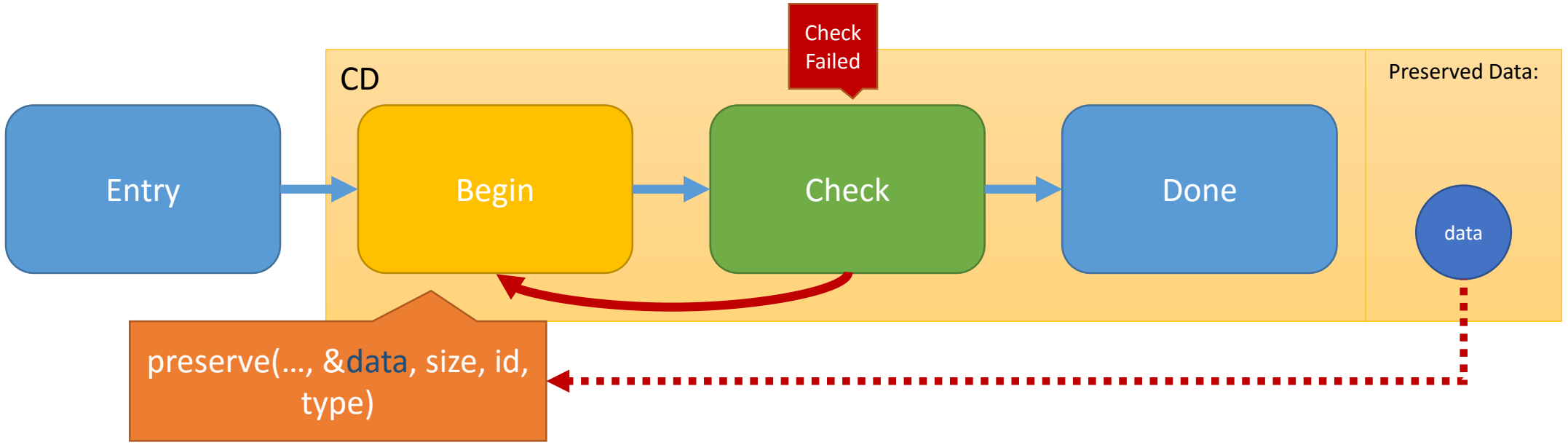
Preservation Semantics

- The first call to swarm_ContainmentDomain_preserve() within a CD with a given ID will *preserve* data.



Preservation Semantics

- Subsequent calls will *restore* the data from preserved memory.

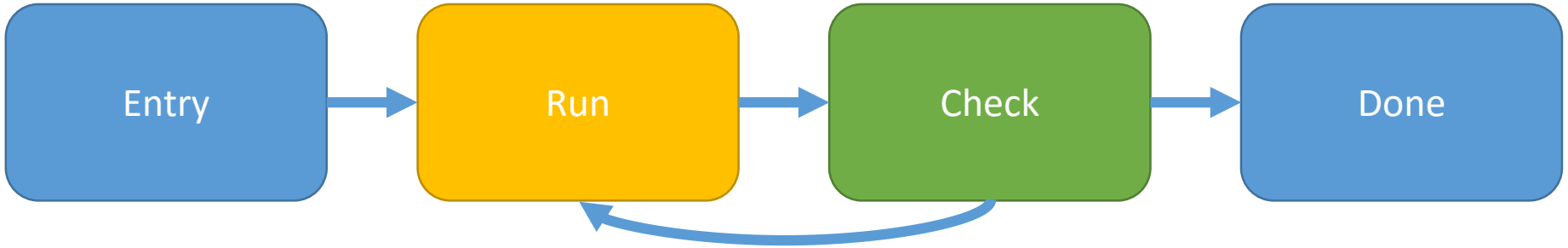


Simple Containment Domain API Example

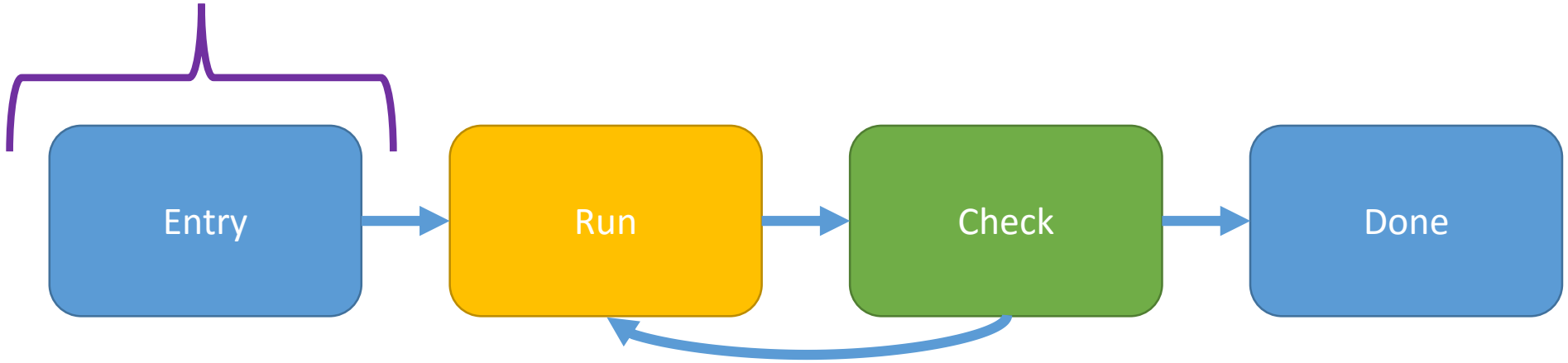
A High Level Glance

- Four codelets involved
  - Entry – Initializes the CD, etc.
  - Run – runs the code
  - Check – performs the check; causes re-execution of run if failed
  - Done – ran if check is successful

Codelet Graph:
A simple example



Codelet Graph:
Execute ‘Entry’



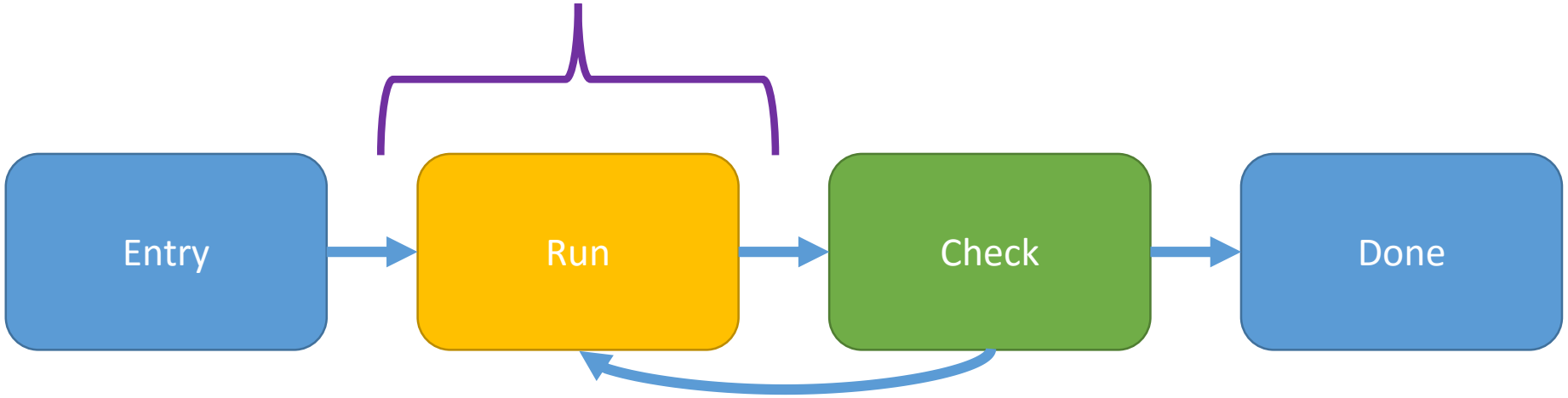
Entry Codelet:

Initialize the CD; specify the codelets involved

```
CODELET_IMPL_BEGIN_NOCANCEL(entry)
   int *C = malloc(sizeof(int));
   mult_t *ctxt = malloc(sizeof(mult_t));
   ctxt->A = &gA;
   ctxt->B = &gB;
   ctxt->C = C;

   swarm_ContainmentDomain_init(&cd);
   swarm_ContainmentDomain_begin(&cd, &CODELET(run), ctxt,
       &CODELET(check), ctxt, &CODELET(done), ctxt);
CODELET_IMPL_END;
```

Codelet Graph:
Execute Run



Run Codelet:

Tell SWARM to preserve input A and B

```
CODELET_IMPL_BEGIN_NOCANCEL(run)
        mult_t *ctxt = THIS;
        swarm_ContainmentDomain_preserve(&cd, &gA, sizeof(int), 0, swarm_CONTAINMENTDOMAIN_COPY );
        swarm_ContainmentDomain_preserve(&cd, &gB, sizeof(int), 1, swarm_CONTAINMENTDOMAIN_COPY );
        *ctxt->C = *ctxt->A * *ctxt->B;
        swarm_dispatch(NEXT, NEXT_THIS, NULL, NULL, NULL);
CODELET_IMPL_END;
```

Run Codelet:
Execute Multiplication

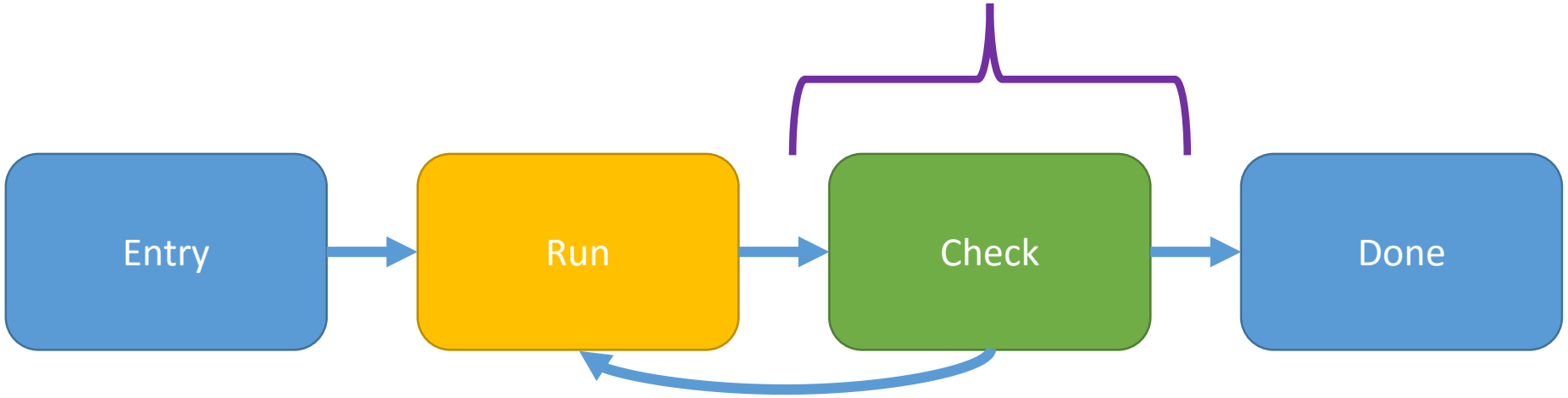
```
CODELET_IMPL_BEGIN_NOCANCEL(run)
        mult_t *ctxt = THIS;
        swarm_ContainmentDomain_preserve(&cd, &gA, sizeof(int), 0, swarm_CONTAINMENTDOMAIN_COPY );
        swarm_ContainmentDomain_preserve(&cd, &gB, sizeof(int), 1, swarm_CONTAINMENTDOMAIN_COPY );
        *ctxt->C = *ctxt->A * *ctxt->B;
        swarm_dispatch(NEXT, NEXT_THIS, NULL, NULL, NULL);
CODELET_IMPL_END;
```

Run Codelet:

Dispatch the next codelet (Check)

```
CODELET_IMPL_BEGIN_NOCANCEL(run)
        mult_t *ctxt = THIS;
        swarm_ContainmentDomain_preserve(&cd, &gA, sizeof(int), 0, swarm_CONTAINMENTDOMAIN_COPY );
        swarm_ContainmentDomain_preserve(&cd, &gB, sizeof(int), 1, swarm_CONTAINMENTDOMAIN_COPY );
        *ctxt->C = *ctxt->A * *ctxt->B;
        swarm_dispatch(NEXT, NEXT_THIS, NULL, NULL, NULL);
CODELET_IMPL_END;
```

Codelet Graph:
Execute Check



Check Codelet:

Re-execute the operation and compare with the initial result computed in the ‘run’ codelet

```
CODELET_IMPL_BEGIN_NOCANCEL(check)
        mult_t *ctxt = THIS;
        swarm_bool_t success = (*ctxt->C == *ctxt->A * *ctxt->B);
        swarm_dispatch(NEXT, NEXT_THIS, (void*)success, NULL, NULL);
CODELET_IMPL_END;
```

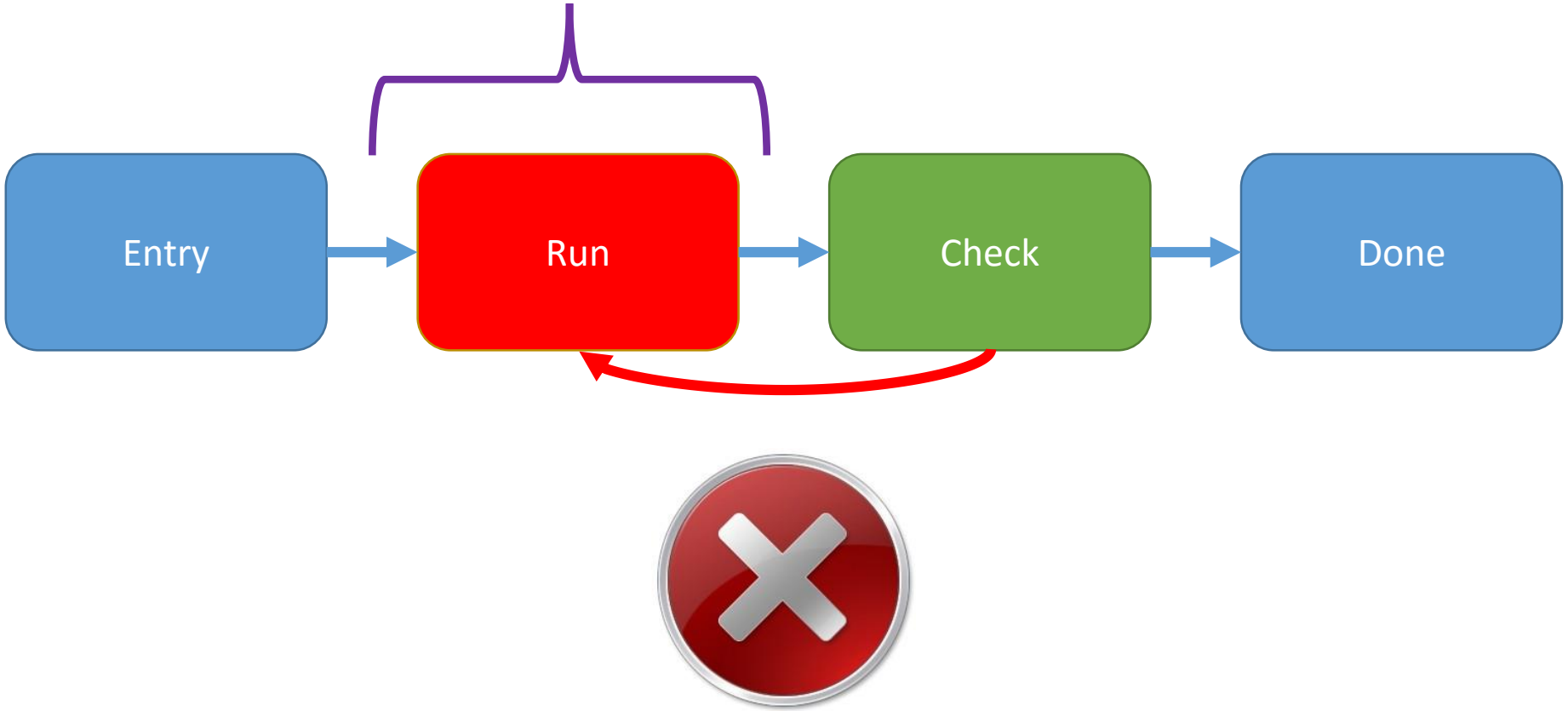
Check Codelet:

Dispatch the next codelet either ‘done’ or ‘run’, dependent on the value of ‘success’

```
CODELET_IMPL_BEGIN_NOCANCEL(check)
        mult_t *ctxt = THIS;
        swarm_bool_t success = (*ctxt->C == *ctxt->A * *ctxt->B);
        swarm_dispatch(NEXT, NEXT_THIS, (void*)success, NULL, NULL);
CODELET_IMPL_END;
```

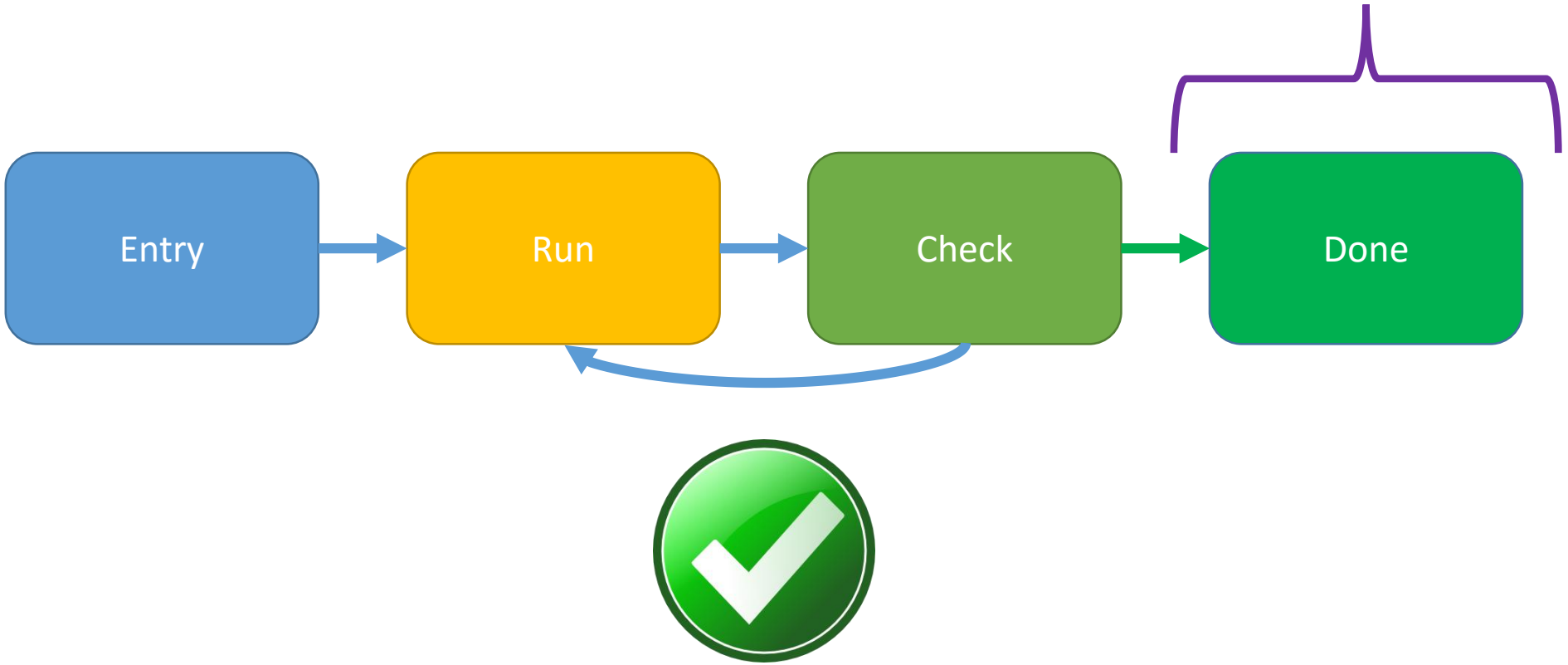
Codelet Graph:

If failure – re-execute ‘Run’

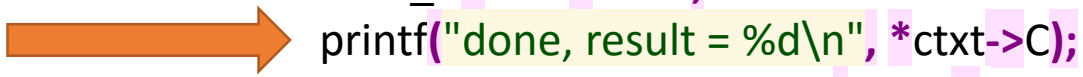


Codelet Graph:

If success – execute ‘Done’



Done Codelet:
Print results



```
CODELET_IMPL_BEGIN_NOCANCEL(done)
        mult_t *ctxt = THIS;
        printf("done, result = %d\n", *ctxt->C);
        swarm_shutdownRuntime(NULL);
CODELET_IMPL_END;
```

Nested CD Example

A simple nested domain example

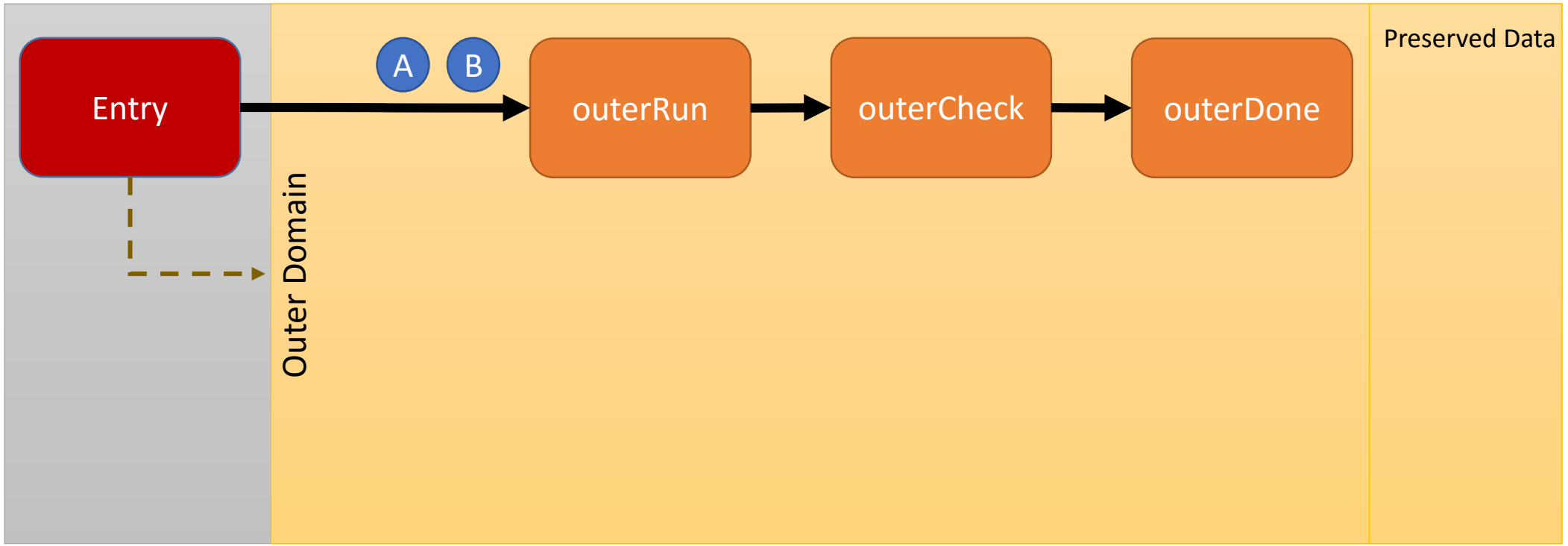
A High Level Glance

- Seven codelets involved
  - Entry – Initializes the outerCD, etc.
  - RunOuter – runs the outer domain code
  - CheckOuter – performs the check; causes re-execution of runOuter if failed
  - DoneOuter – ran if check for the outer domain is successful
  - RunInner – runs the inner domain code
  - CheckInner – performs the check; causes re-execution of runInner if failed
  - DoneInner – ran if the check for the inner domain is successful

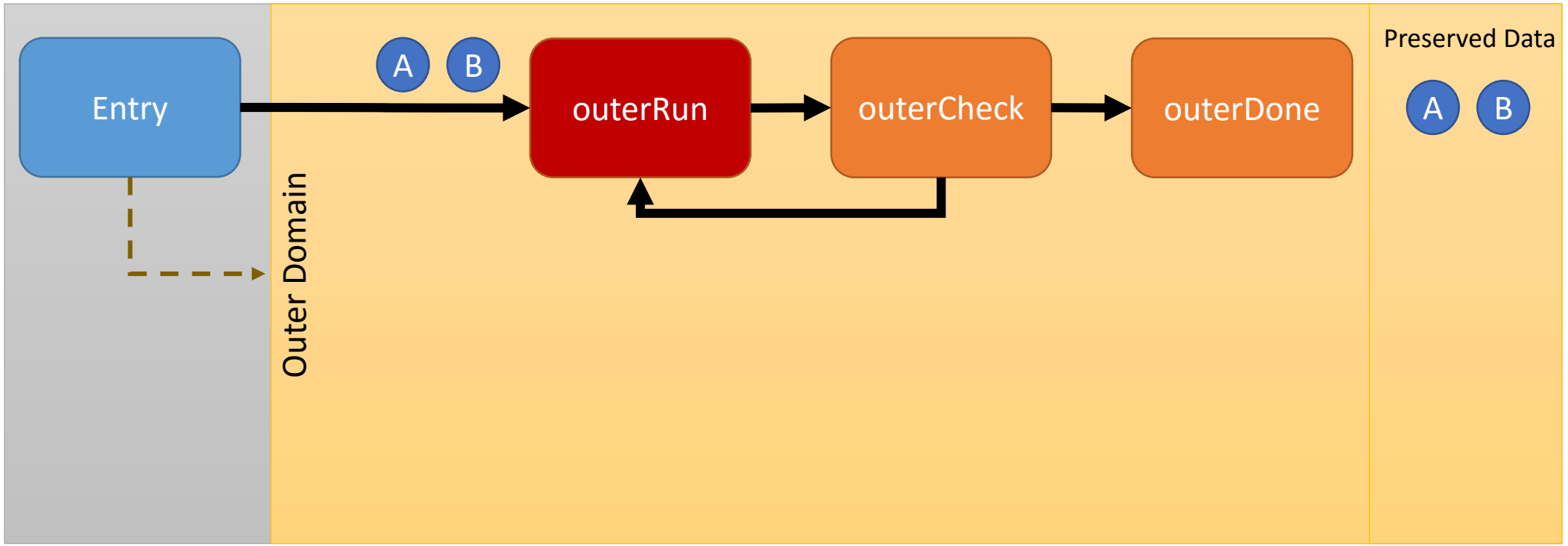
Codelet Graph:
Entry codelet starts

Codelet Graph:

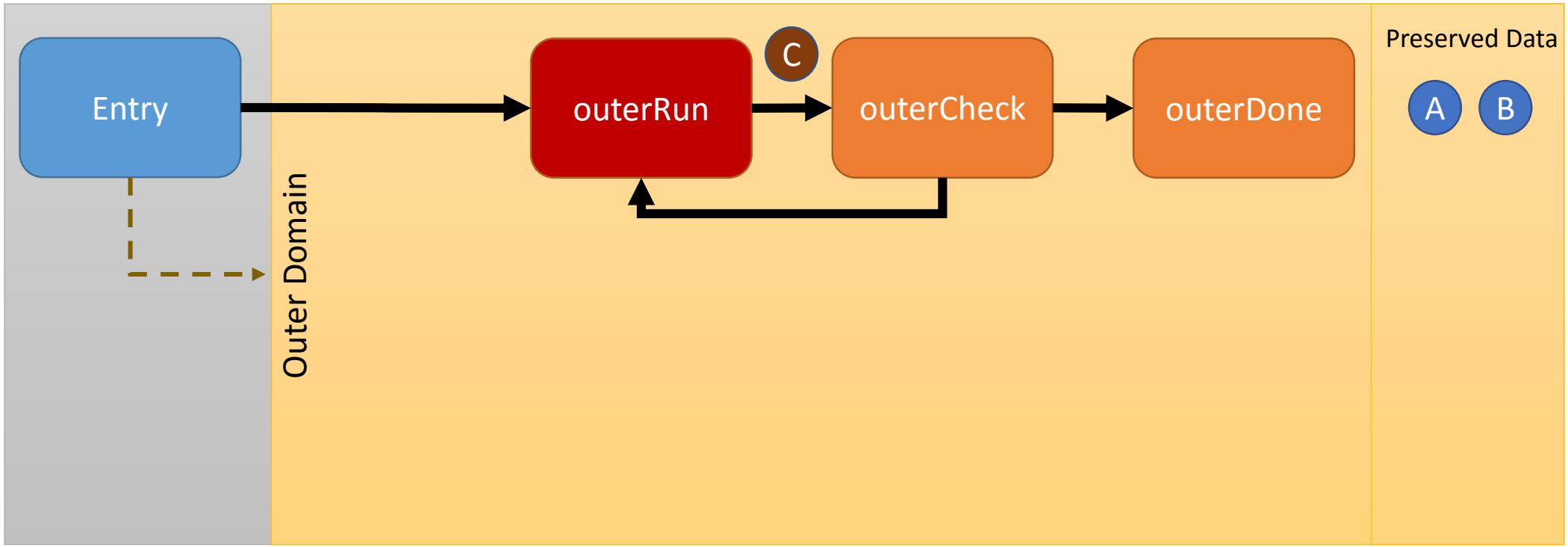
Entry codelet creates the outer domain with inputs A & B



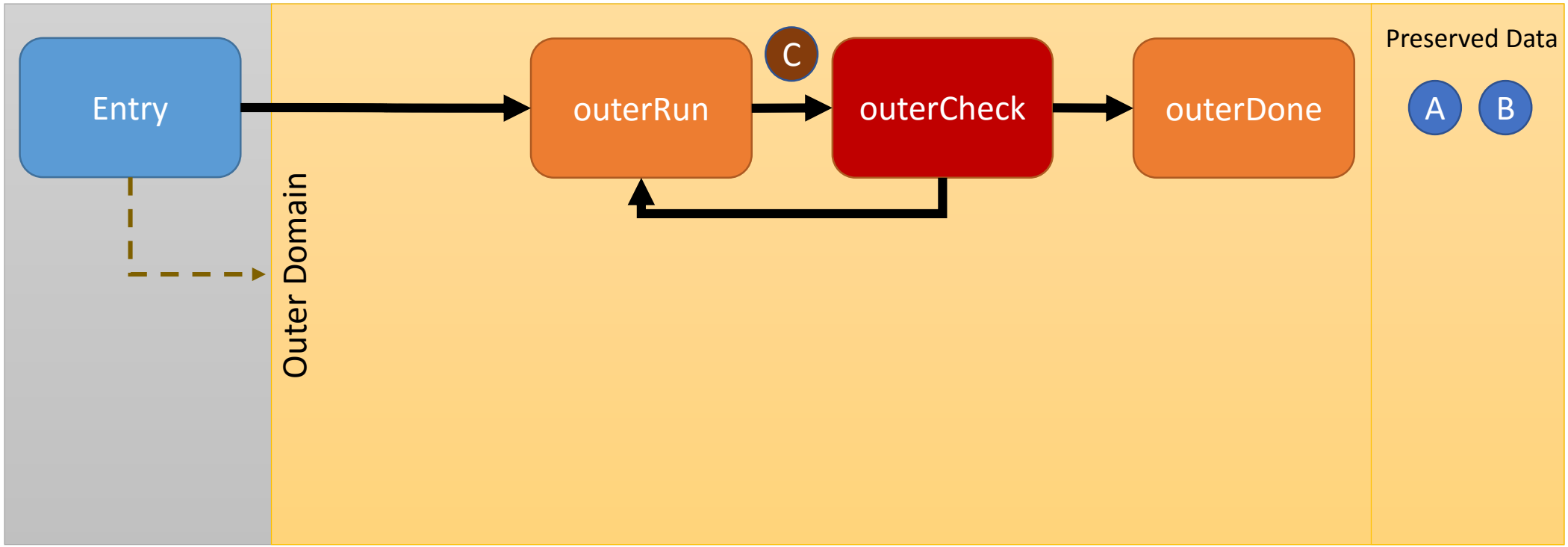
Codelet Graph:
outerRun preserves inputs A & B



Codelet Graph:
outerRun executes C=A*B

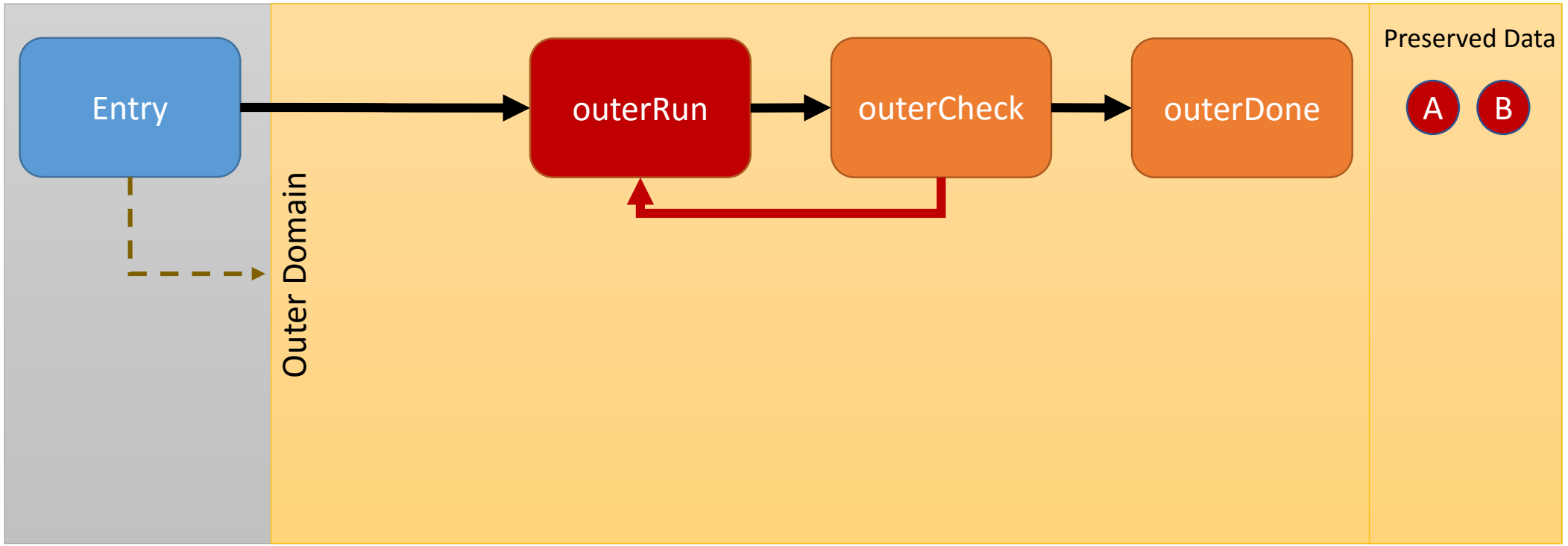


Codelet Graph:
outerCheck verifies the correctness of C

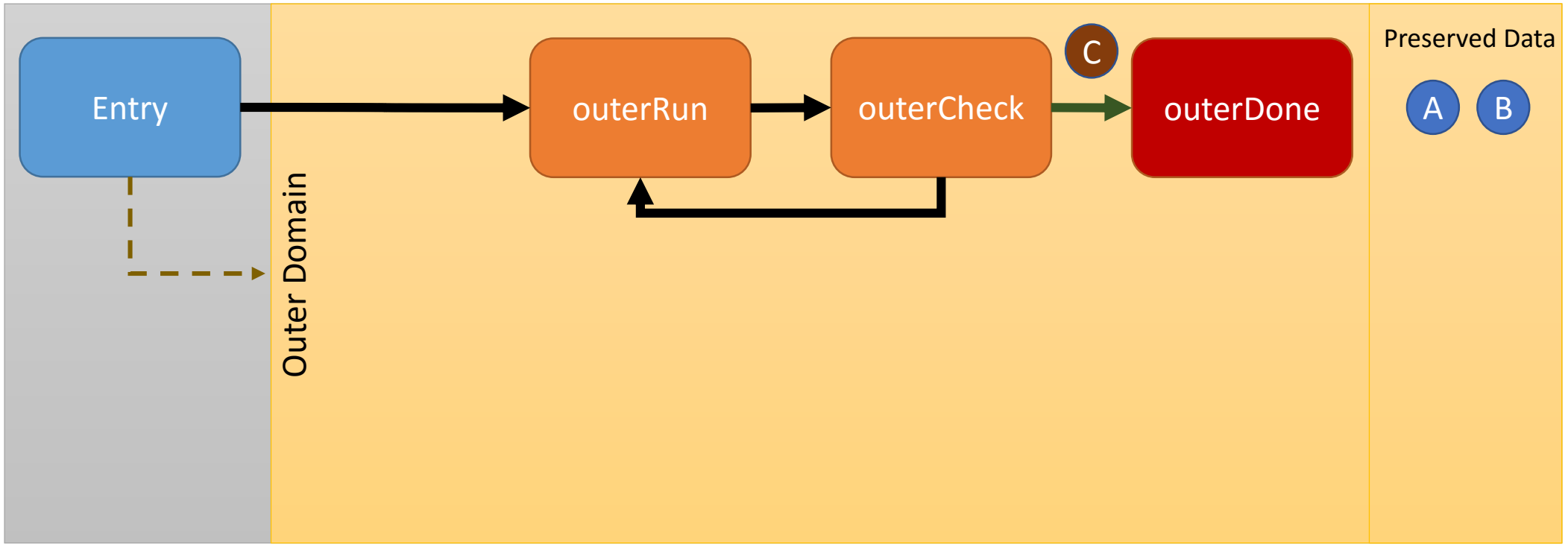


Codelet Graph:

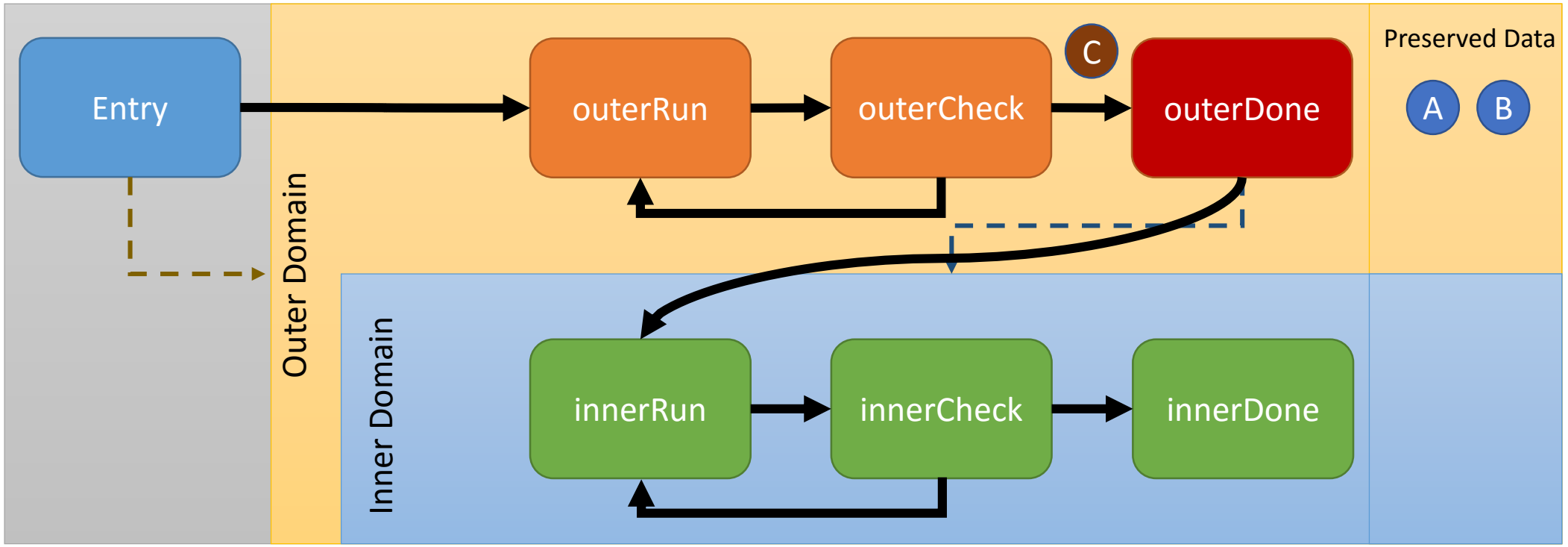
if the result is incorrect, outerRun is executed again using the preserved A & B



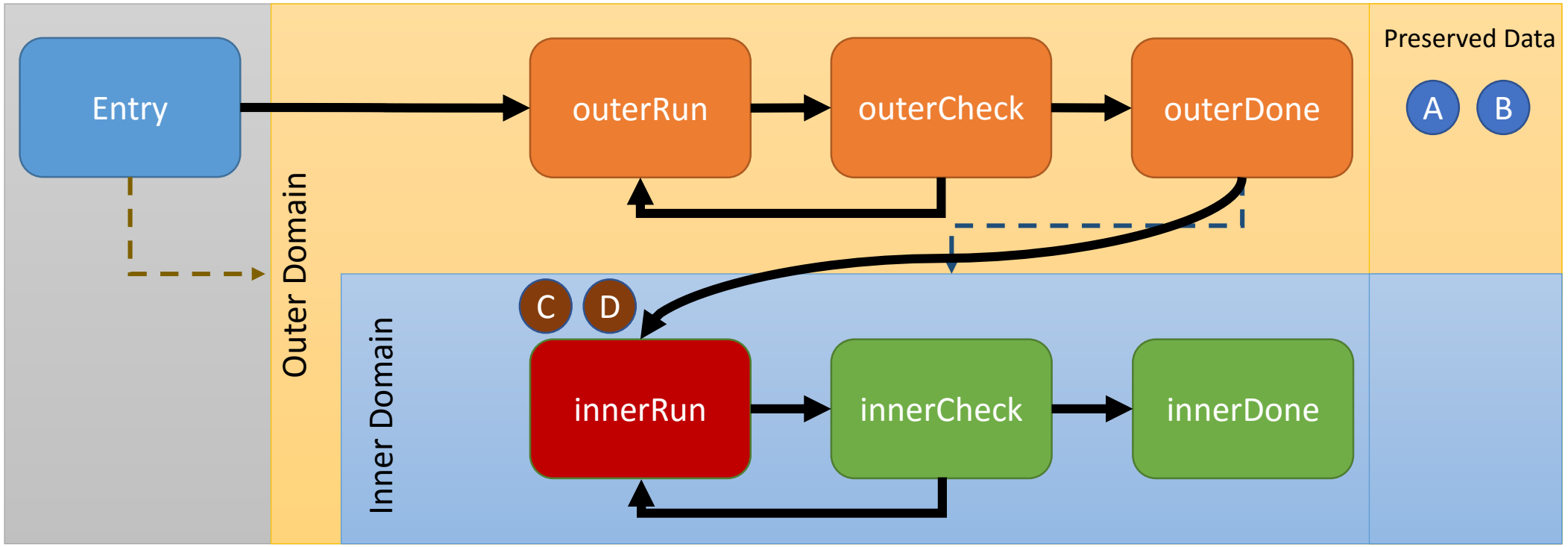
Codelet Graph:
if the result is correct, outerDone is executed



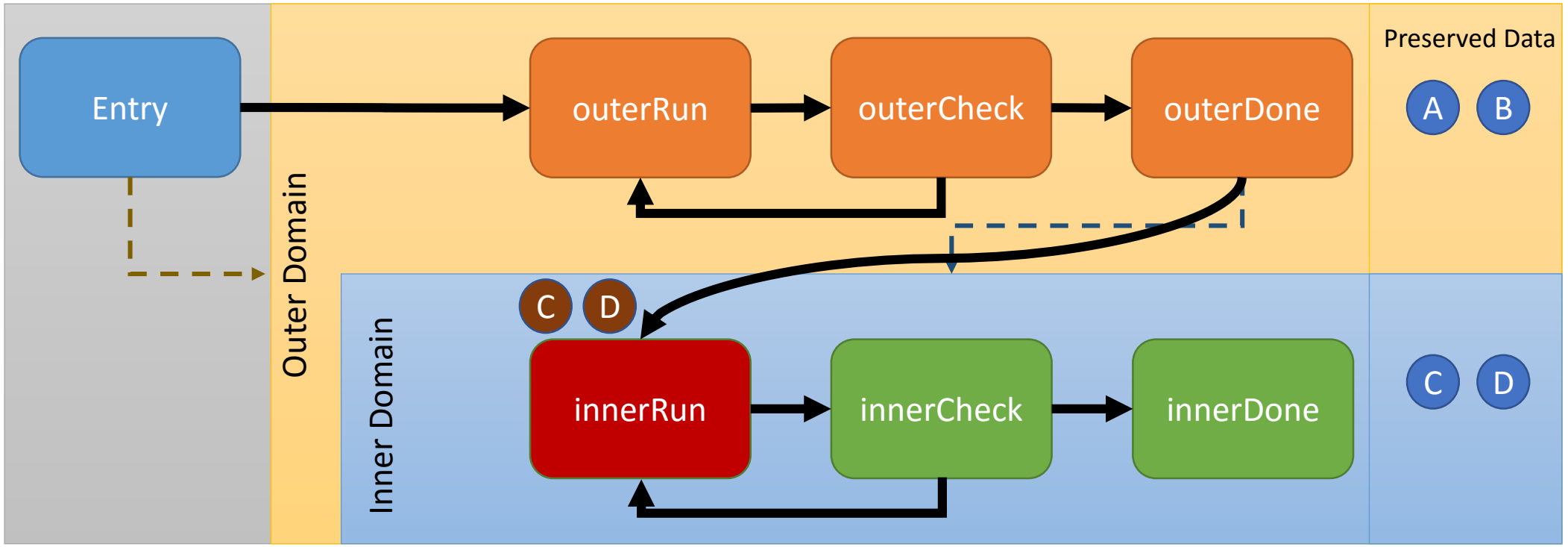
Codelet Graph:
outerDone creates the inner domain



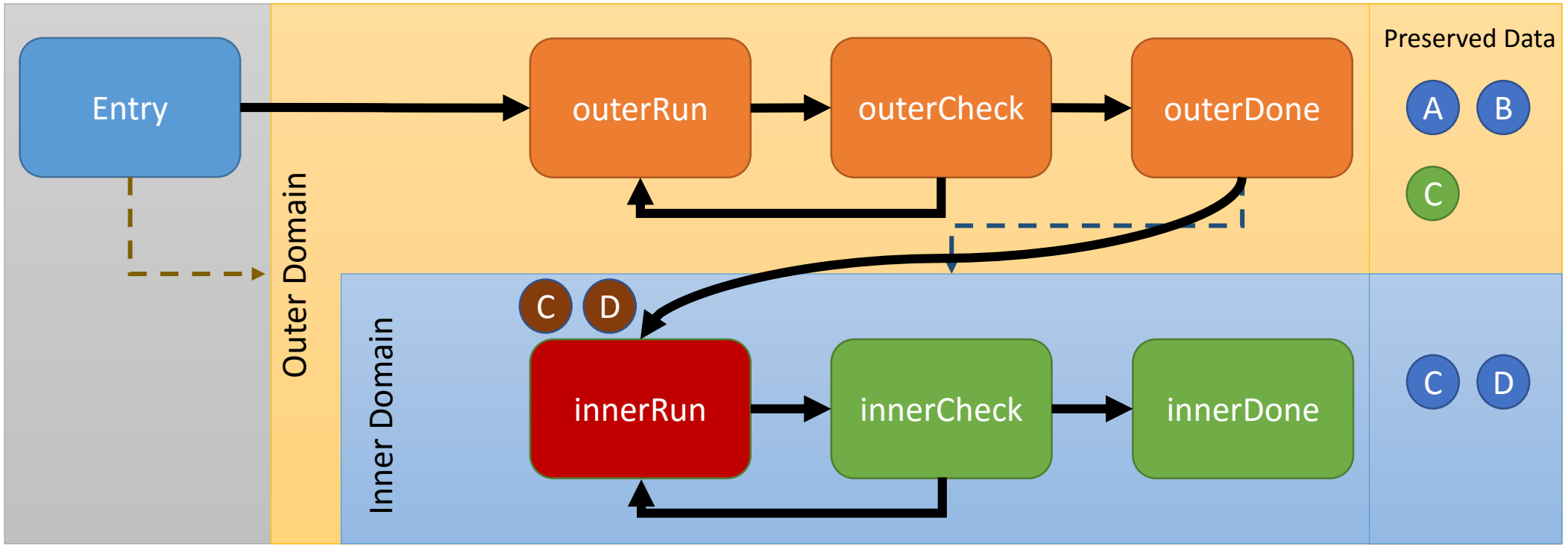
Codelet Graph:
innerRun is executed with inputs C & D



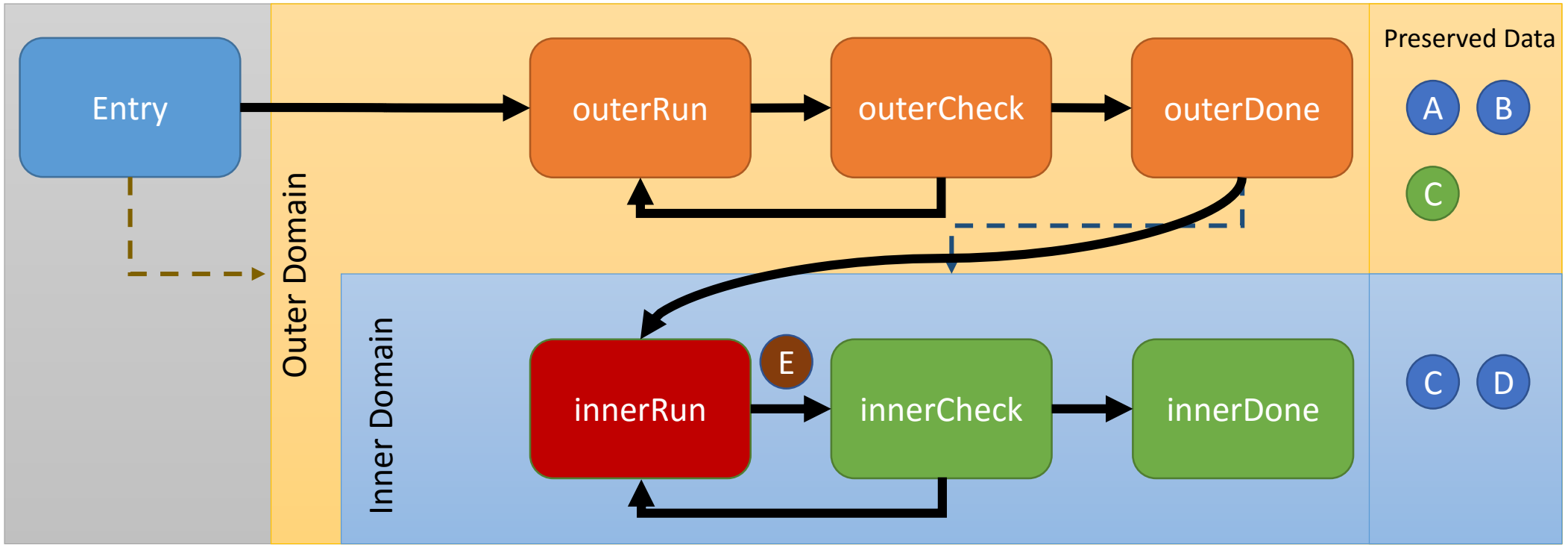
Codelet Graph:
innerRun preserves C & D in the inner domain



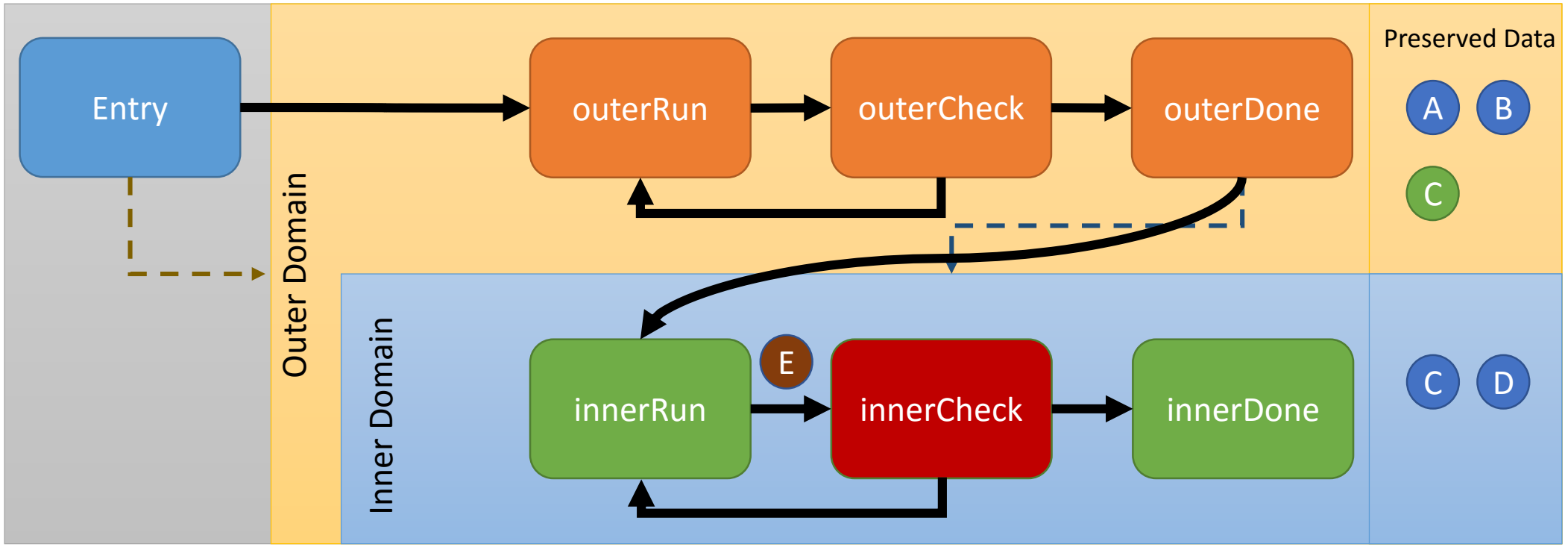
Codelet Graph:
innerRun preserves C in the outer domain



Codelet Graph:
innerRun executes E=C+D

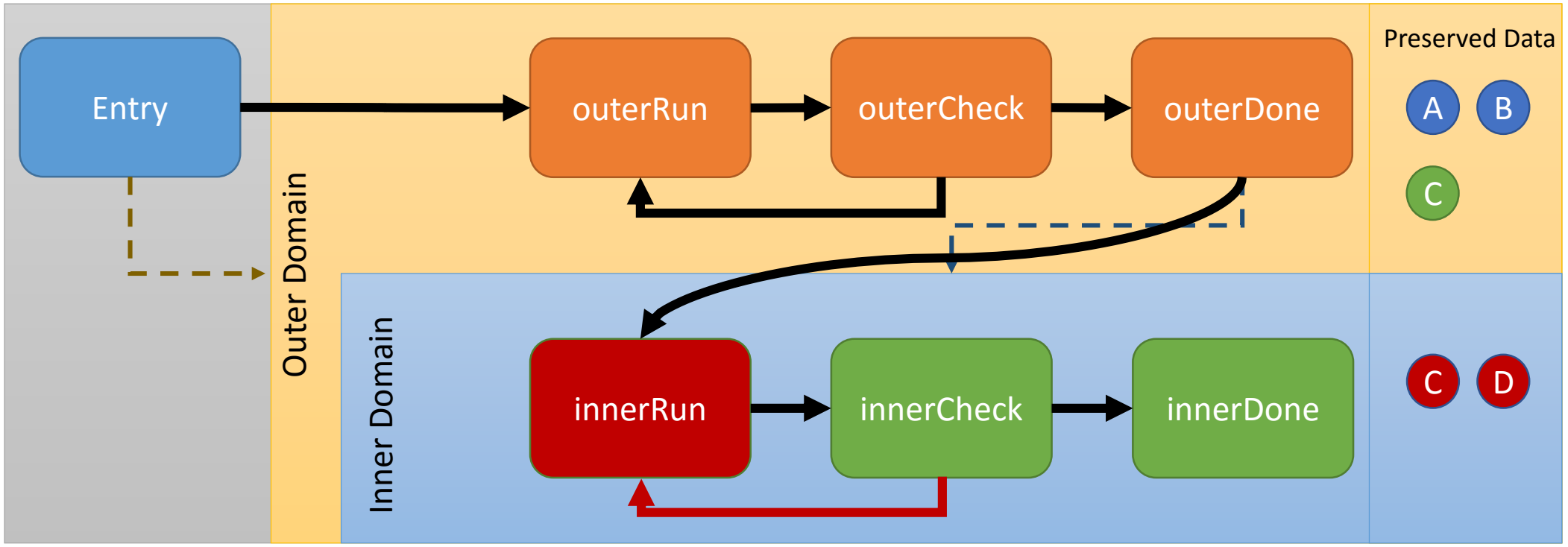


Codelet Graph:
outerCheck verifies the correctness of E



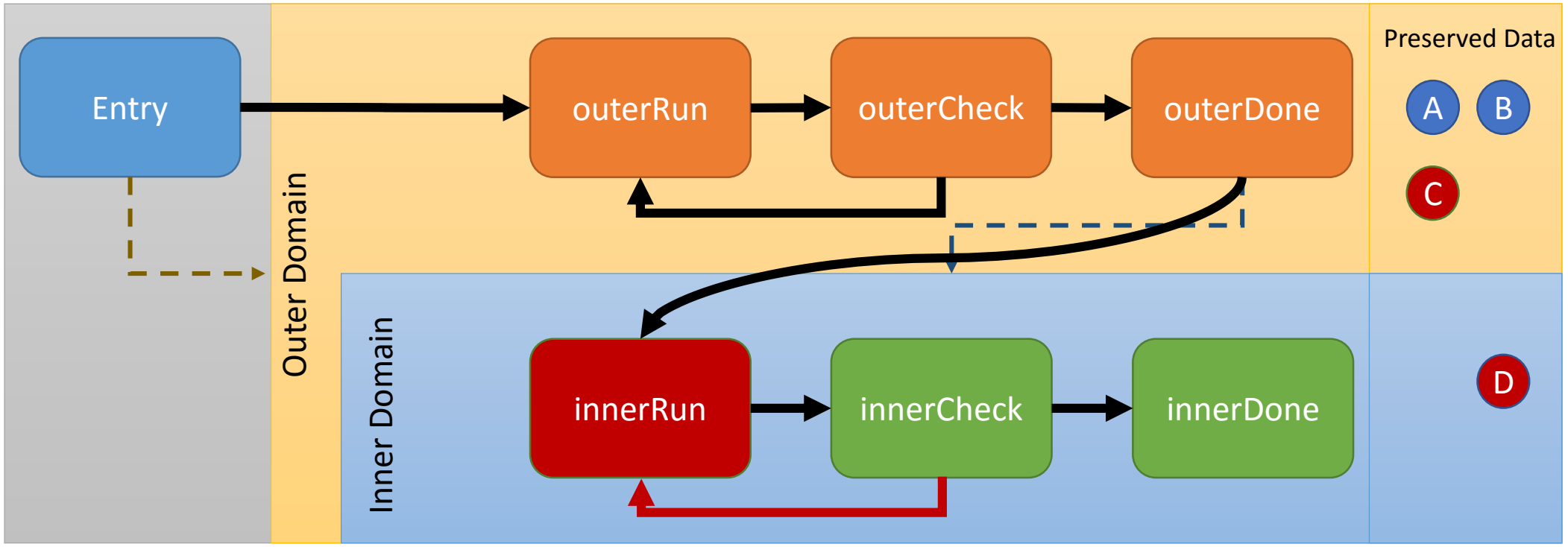
Codelet Graph:

if the result is incorrect and C&D are available, innerRun is re-executed



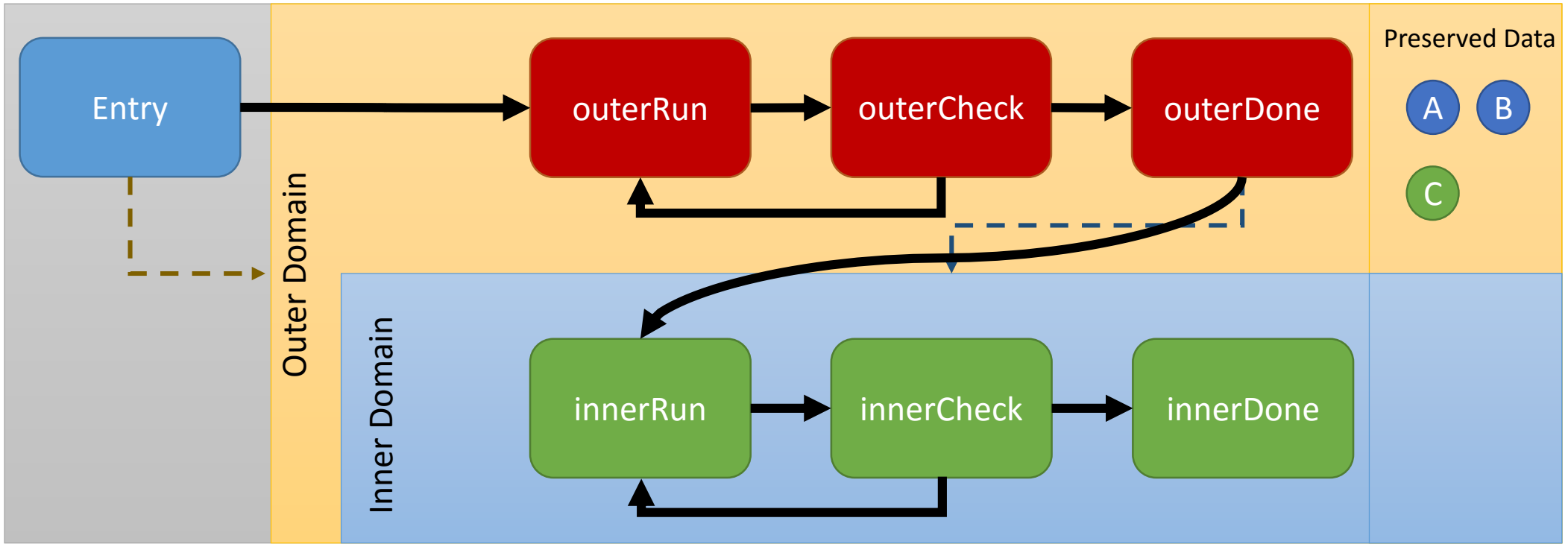
Codelet Graph:

if C becomes unavailable in the inner domain, it is retrieved for the outer domain



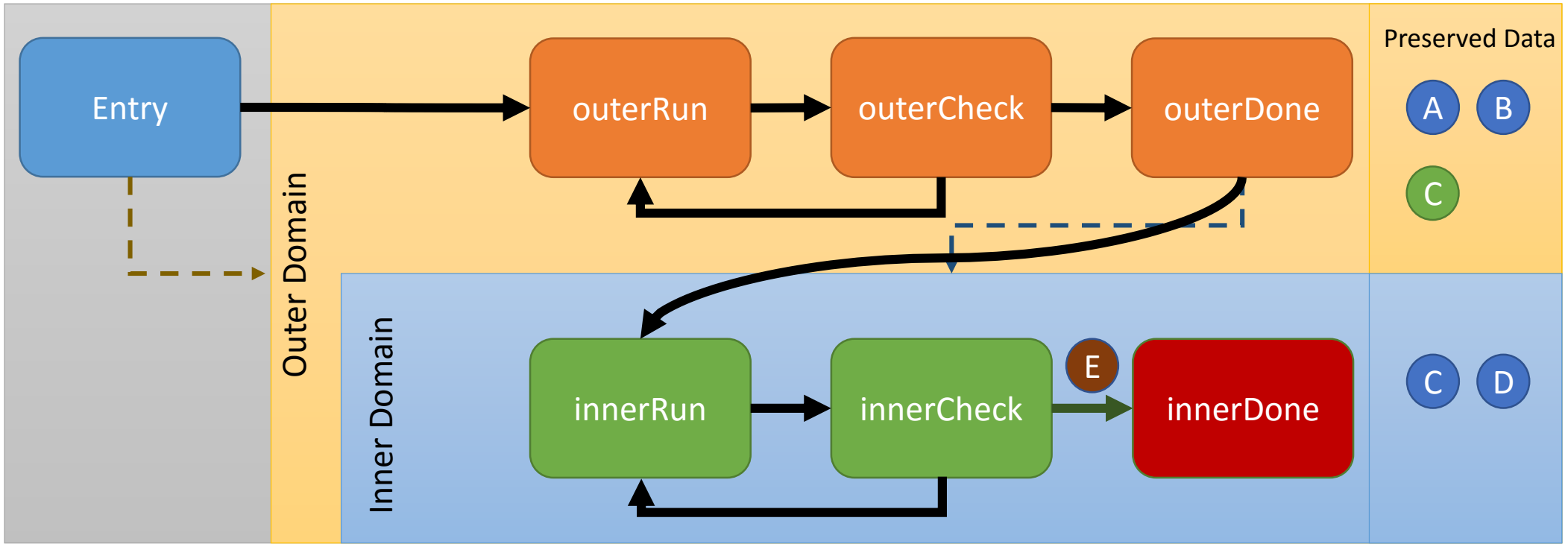
Codelet Graph:

if D becomes unavailable in the inner domain, the outer domain is re-executed using the preserved inputs and then the inner domain is run again



Codelet Graph:

If innerCheck succeeds, then innerDone is executed and the program finishes



Conclusion and Summary

- We present containment domains and their advantages compared standard resiliency techniques.
- We provided two examples of how to implement containment domains using SWARM.
- As ongoing work we are doing a back-of-the-envelope analysis to determine where to inject preservation statements for containment domains.
- We recently submitted an extended abstract for publication to the Mini-Symposium on Energy and Resilience in Parallel Programming (ERPP 2015) to be held in conjunction with International Conference on Parallel Computing (ParCo 2015).