TÉCNICO LISBOA

1st Mini-Project: File Transfer

Reliable Data Transfer

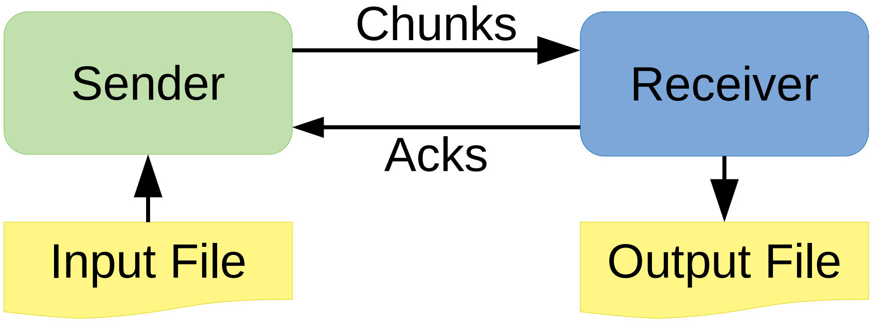
Overview

What you’ll learn:

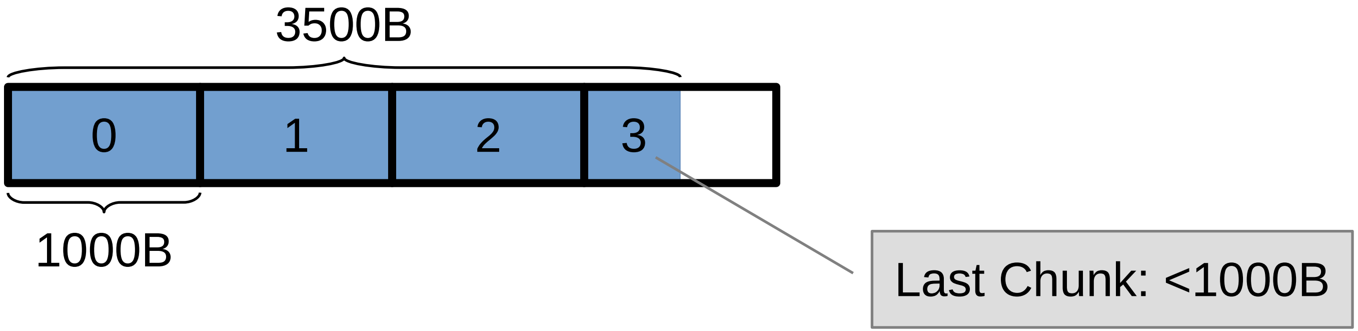
- Reliable data xfer
- UDP sockets

Create file transfer system

- **File Sender**
- **File Receiver**

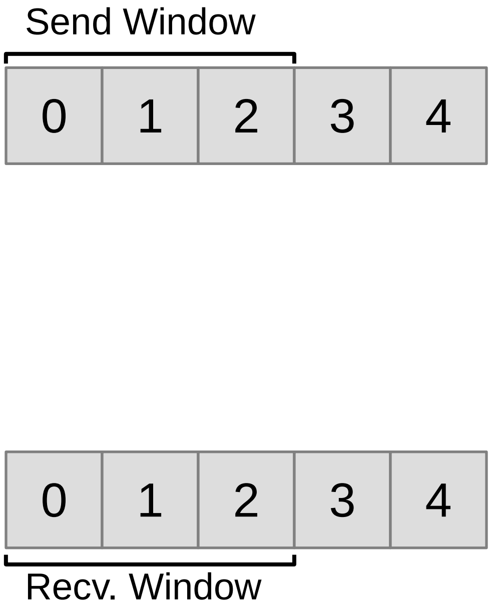


Overview: Files to Chunks



```
typedef struct __attribute__((__packed__)) data_pkt_t {
  uint32_t seq_num;
  char data[1000];
} data_pkt_t;

typedef struct __attribute__((__packed__)) ack_pkt_t {
  uint32_t seq_num;
  uint32_t selective_acks;
} ack_pkt_t;
```

Overview: Reliable Data Transfer



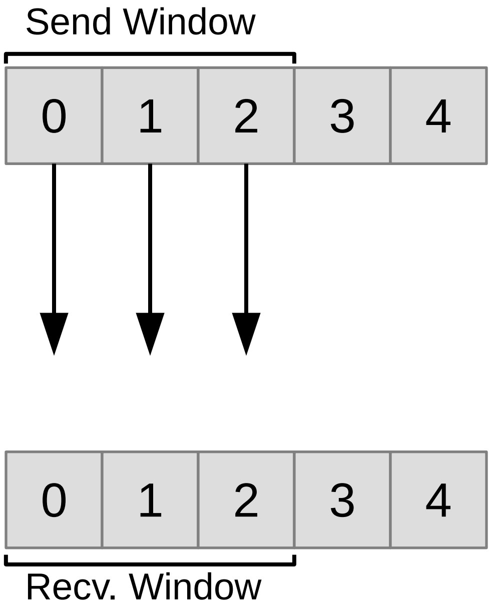
Send Window

0	1	2	3	4

0	1	2	3	4

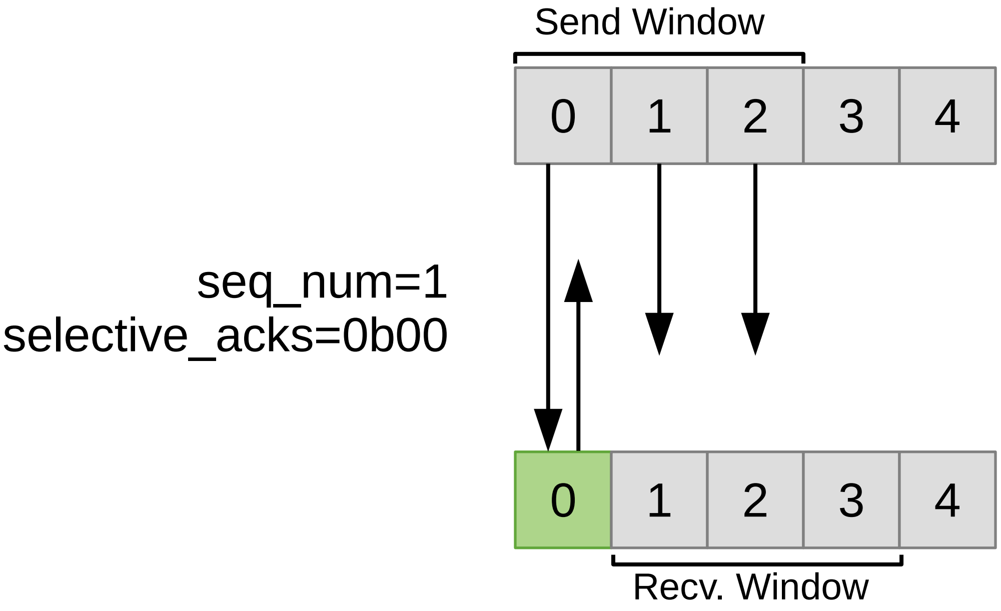
Recv. Window

Overview: Reliable Data Transfer

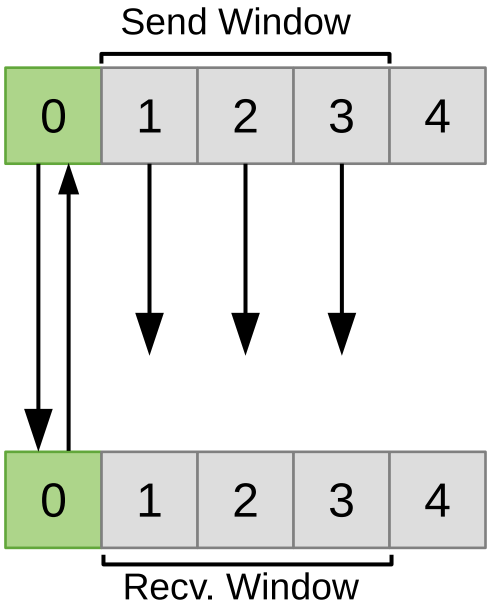


seq_num=0,1,2

Overview: Reliable Data Transfer



Overview: Reliable Data Transfer



Send Window

0	1	2	3	4

seq_num=3

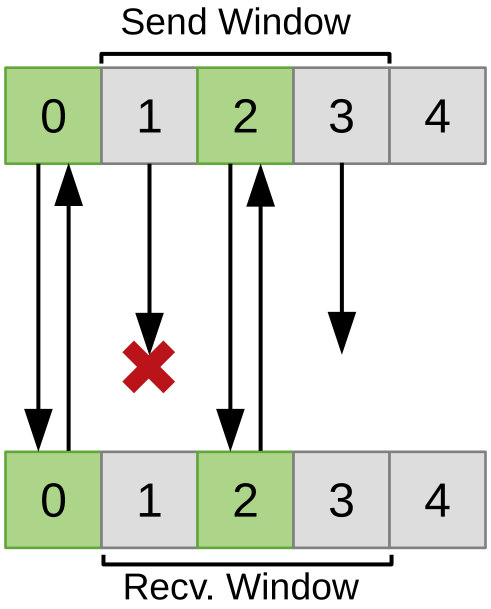
0	1	2	3	4

Recv. Window

Overview: Reliable Data Transfer

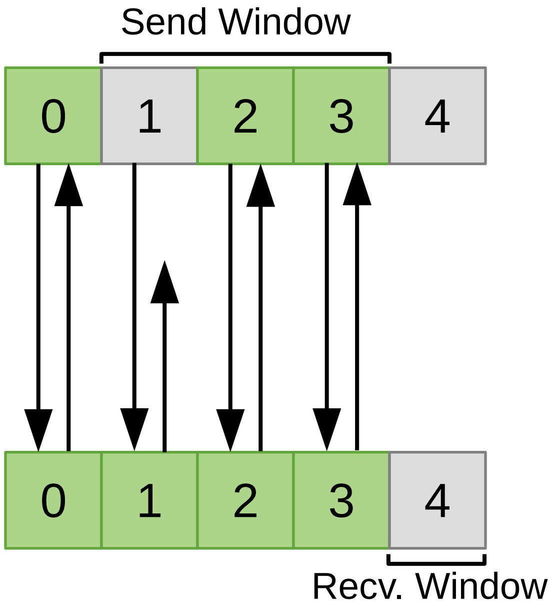
seq_num=1
selective_acks=0b01

DupAck → Retransmit 1

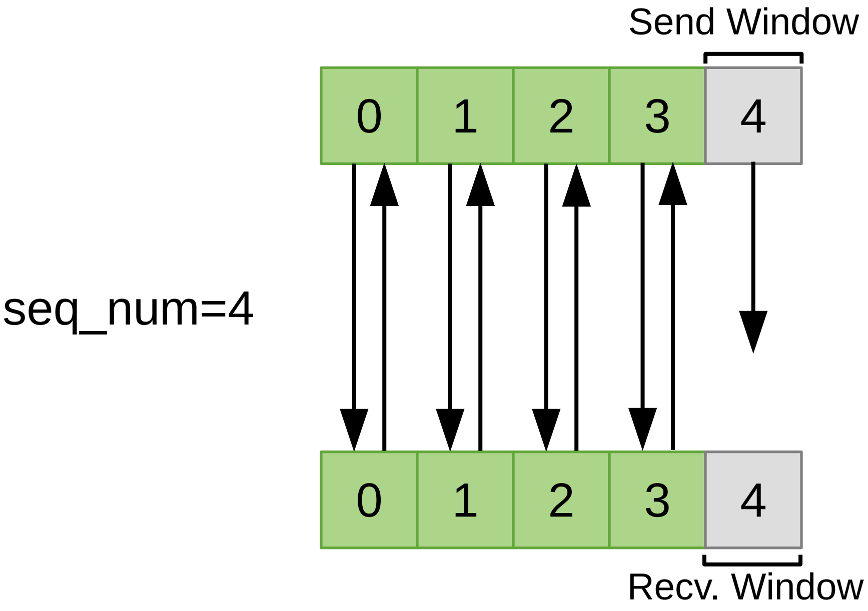


Overview: Reliable Data Transfer

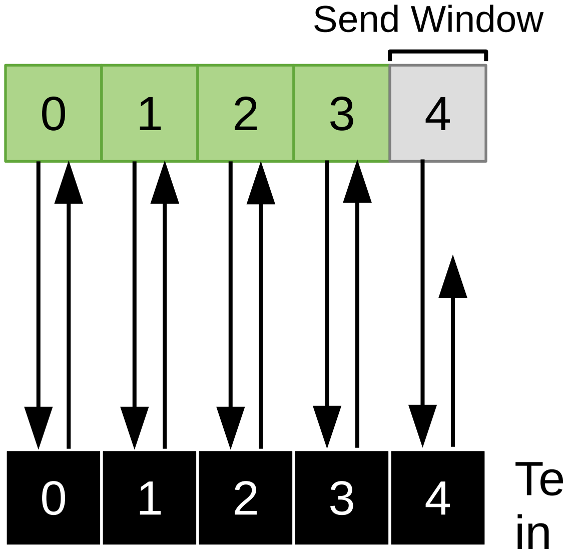
seq_num=4
selective_acks=0b00



Overview: Reliable Data Transfer



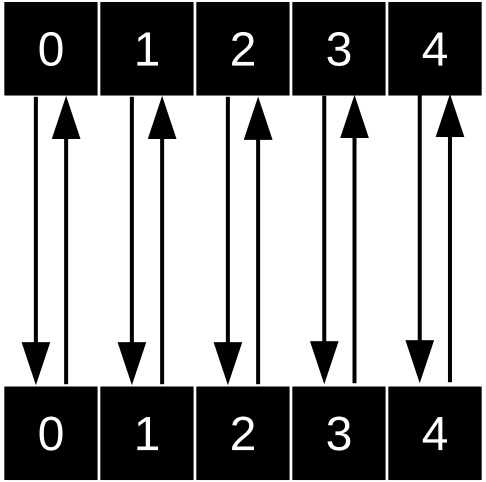
Overview: Reliable Data Transfer



seq_num=5
selective_acks=0b00

Terminates 4s later,
in case ACK lost.

Overview: Reliable Data Transfer



Terminates immediately.

Terminates 4s later,
in case ACK lost.

Overview: Reliable Data Transfer

Notes

- Chunks start at 0
- Fields in network order
  - Use htonl(), ntohl()
- Ack = recv. window
  - seq_num = base
  - selective_acks skips first (will always be 0)

RDT Modes and Window Size

- Stop-and-Wait
  - Send = 1, Receive = 1
- Go-Back-N
  - Send = N, Receive = 1
- Selective Repeat
  - Send = N, Receive = M <= N

Submission

- Develop your code on: https://gitlab.rnl.tecnico.ulisboa.pt
- Include:
  - Code
  - Makefile in base folder
  - No build artifacts
- Tag submission as project1-submission:

```
:~$ git tag project1-submission
:~$ git push origin project1-submission
```

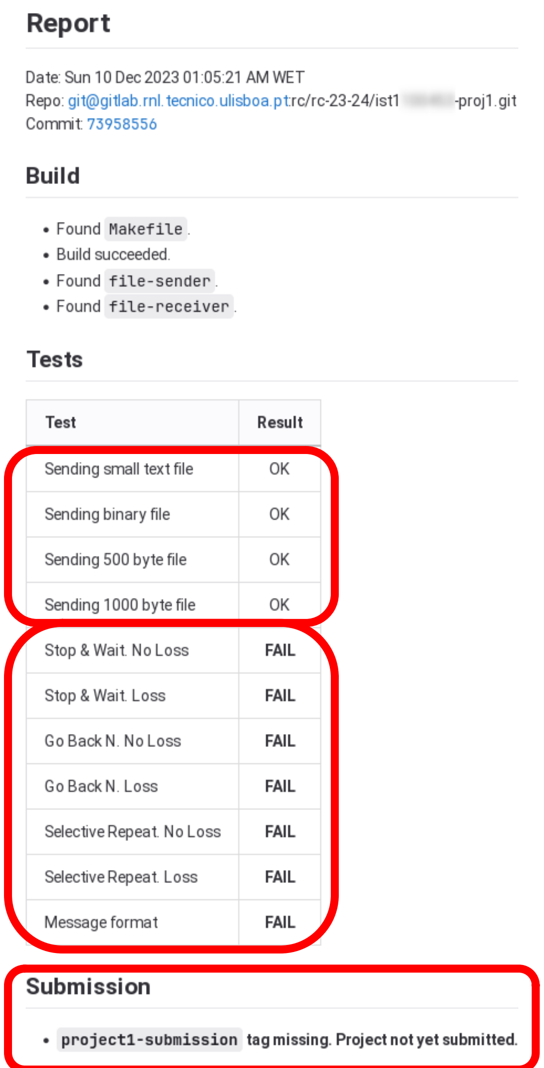
- Must build with **`make`**
  - Generate file-sender & file-receiver

```
:~$ git clone <repo URL> .

:~$ git checkout project1-submission

:~$ ls
Makefile file-receiver.c file-sender.c

:~$ make

:~$ ls
Makefile file-receiver.c file-receiver
file-sender.c file-sender
```

Automatic Tests

- Nightly builds
  - Simple tests – does not preclude running your own
  - Run on **main** branch and generate **build-report.md**
    - Don’t forget to pull
  - On request: must **delete** report and push to rerun next time
    - Tests will not run if **build-report.md** is found in your repo.

Automatic Tests



Report

Date: Sun 10 Dec 2023 01:05:21 AM WET
Repo: git@gitlab.rnl.tecnico.ulisboa.pt:rc/rc-23-24/ist1[redacted]-proj1.git
Commit: 73958556

Build

- Found `Makefile`.
- Build succeeded.
- Found `file-sender`.
- Found `file-receiver`.

Tests

Test	Result
Sending small text file	OK
Sending binary file	OK
Sending 500 byte file	OK
Sending 1000 byte file	OK
Stop & Wait. No Loss	**FAIL**
Stop & Wait. Loss	**FAIL**
Go Back N. No Loss	**FAIL**
Go Back N. Loss	**FAIL**
Selective Repeat. No Loss	**FAIL**
Selective Repeat. Loss	**FAIL**
Message format	**FAIL**

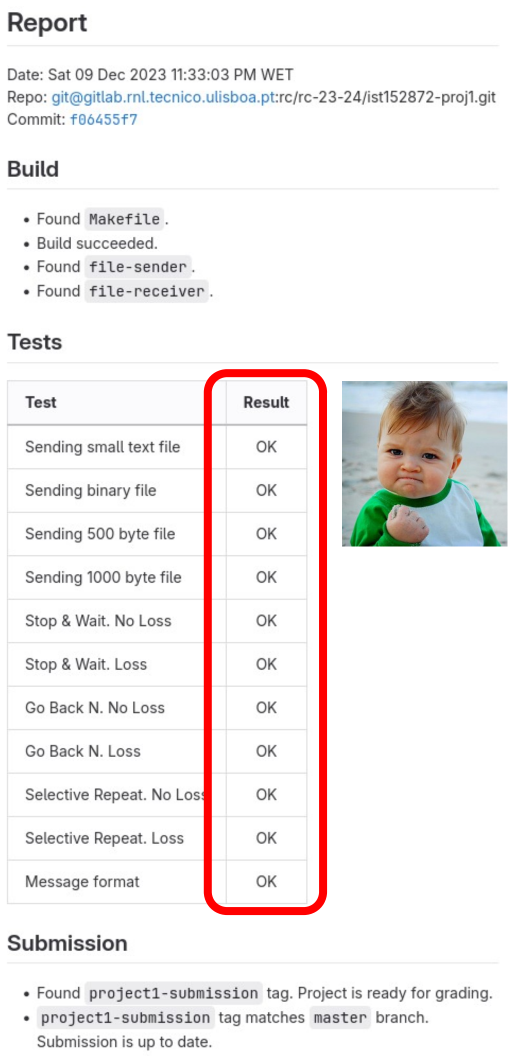
Submission

- `project1-submission` **tag missing. Project not yet submitted.**

Non-reliable transfer works out of the box.

Very basic tests. Run your own tests!

Don't forget to submit!



Report

Date: Sat 09 Dec 2023 11:33:03 PM WET
Repo: git@gitlab.rnl.tecnico.ulisboa.pt:rc/rc-23-24/ist152872-proj1.git
Commit: f06455f7

Build

- Found `Makefile`.
- Build succeeded.
- Found `file-sender`.
- Found `file-receiver`.

Tests

Test	Result
Sending small text file	OK
Sending binary file	OK
Sending 500 byte file	OK
Sending 1000 byte file	OK
Stop & Wait. No Loss	OK
Stop & Wait. Loss	OK
Go Back N. No Loss	OK
Go Back N. Loss	OK
Selective Repeat. No Loss	OK
Selective Repeat. Loss	OK
Message format	OK

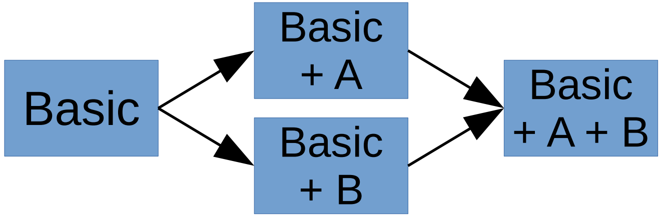
Submission

- Found `project1-submission` tag. Project is ready for grading.
- `project1-submission` tag matches `master` branch. Submission is up to date.

GIT Primer

- Git is a distributed version control system
  - Tracks versions of code
  - Tracks/merges branches
  - Ubiquitous

- Creates a version graph with branches diverging and merging.

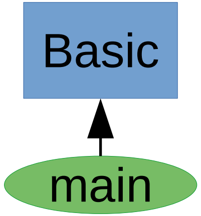


- Synchronizes a local repo with a remote repo.

GIT Primer – Walk-through

```
:~$
```

Remote Repo:

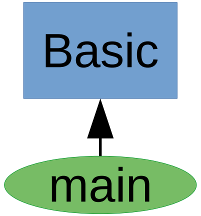


Local Repo:

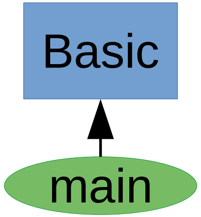
GIT Primer – Walk-through

```
:~$ git clone <url>
```

Remote Repo:



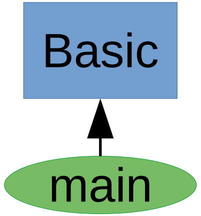
Local Repo:



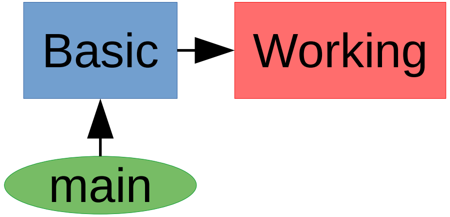
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git status
Untracked files: A.c

:~$ git add A.c
:~$ git status
Changes to be committed:
        new file:   A.c
```

Remote Repo:



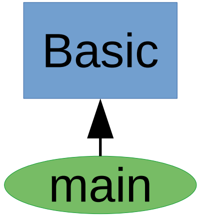
Local Repo:



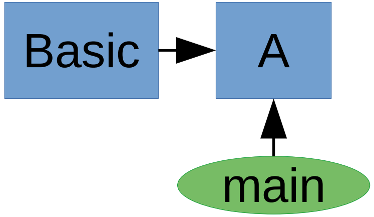
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git add A.c
:~$ git commit -m “Did A”
:~$ git status
# ahead of 'origin/main' by 1 commit.
nothing to commit
```

Remote Repo:



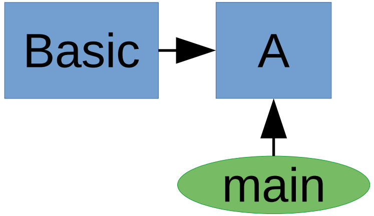
Local Repo:



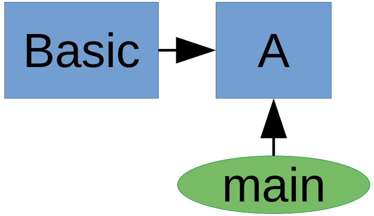
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git add A.c
:~$ git commit -m “Did A”
:~$ git push
```

Remote Repo:



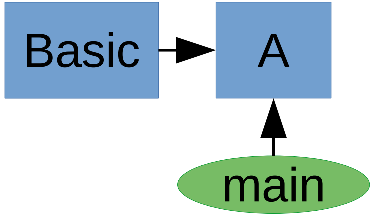
Local Repo:



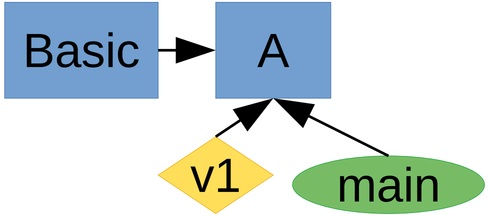
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git add A.c
:~$ git commit -m “Did A”
:~$ git push
:~$ git tag v1
```

Remote Repo:



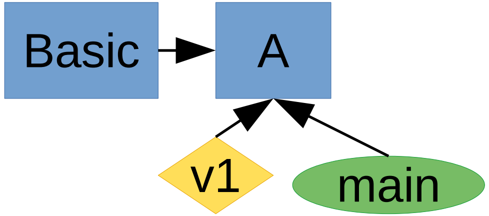
Local Repo:



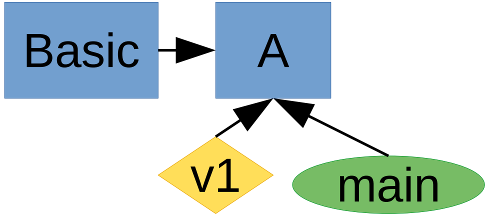
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git add A.c
:~$ git commit -m “Did A”
:~$ git push
:~$ git tag v1
:~$ git push origin v1
```

Remote Repo:



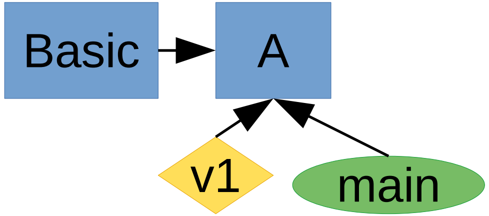
Local Repo:



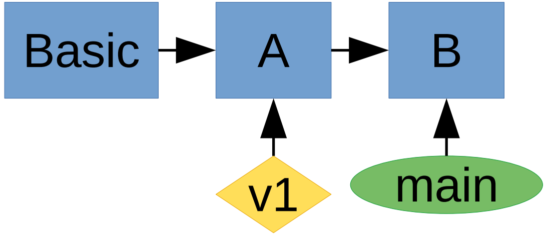
GIT Primer – Walk-through

```
:~$ git clone <url>
:~$ echo stuff > A.c
:~$ git add A.c
:~$ git commit -m “Did A”
:~$ git push
:~$ git tag v1
:~$ git push origin v1
:~$ echo stuff > B.c
:~$ git add B.c
:~$ git commit -m “Did B”
```

Remote Repo:



Local Repo:



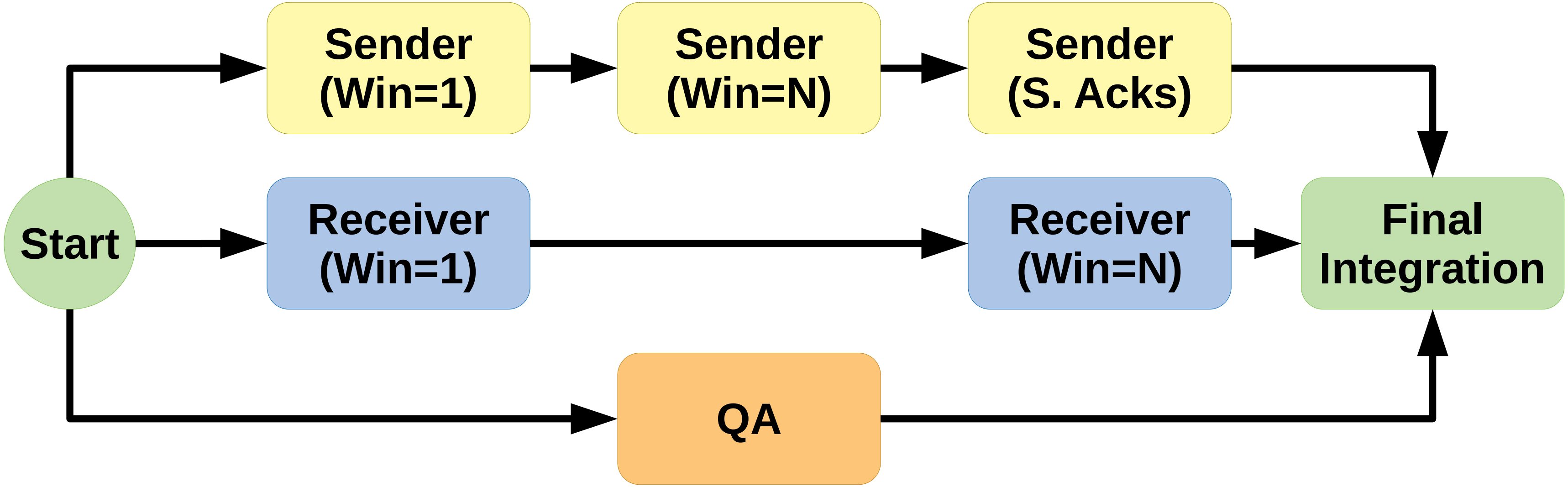
GIT Primer – More Info

- Quick reference: `git help <command>`
- Cheat sheet: https://about.gitlab.com/images/press/git-cheat-sheet.pdf
- Branching and Merging: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
- Full Docs: https://git-scm.com/doc

Advice: Debugging

- Standard output/error will be ignored during grading
  - **`printf(…)`**
- Debug tools also available
  - **log-packets.c**: Packet logging & fault injection
  - **generate-msc.sh**: Log analysis & MSC generation (uses mscgen package)
- Testing
  - Look into **run.sh** for ideas.

Advice: Task Breakdown



Advice: MSC Generation

```
gcc -shared -fPIC -Wall -O0 -g \
    -o log-packets.so log-packets.c -ldl
```

```
LD_PRELOAD = "./log-packets.so" \
    SEND_DELAY="500" \
    DROP_PATTERN="01" \
    PACKET_LOG="sender.log" \
    ./file-sender …
```

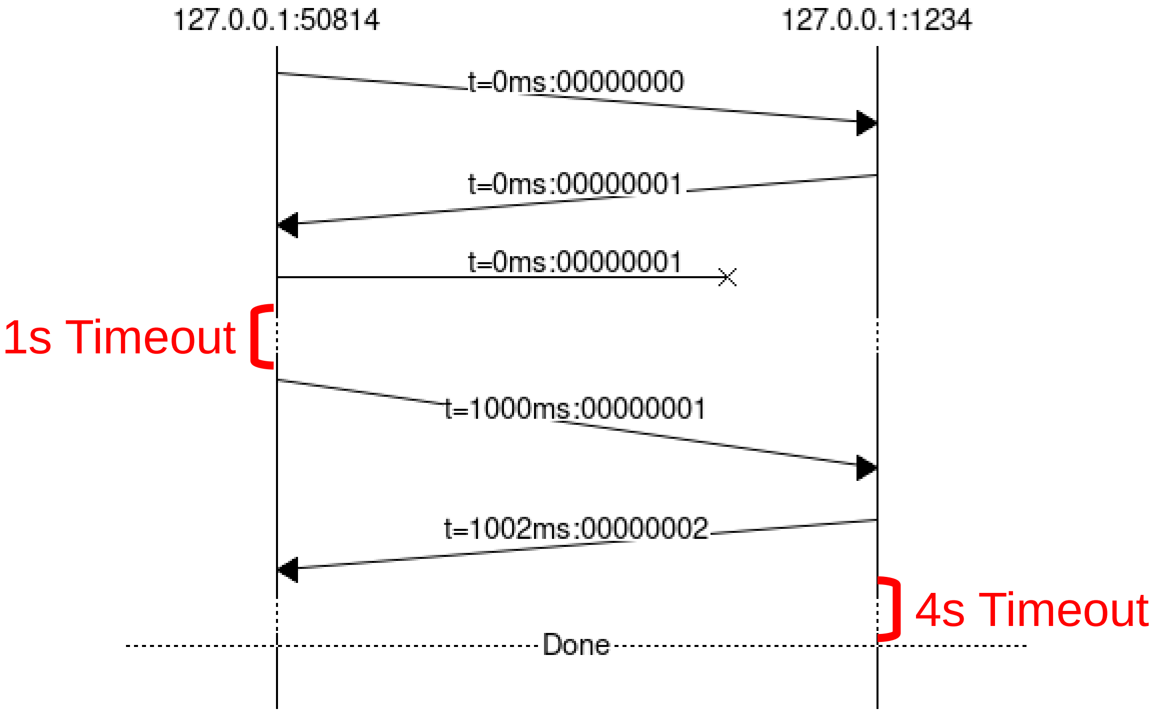
```
./generate-msc.sh msc.eps sender.log receiver.log
```

See: run.sh

Advice: MSCs

Stop-and-Wait

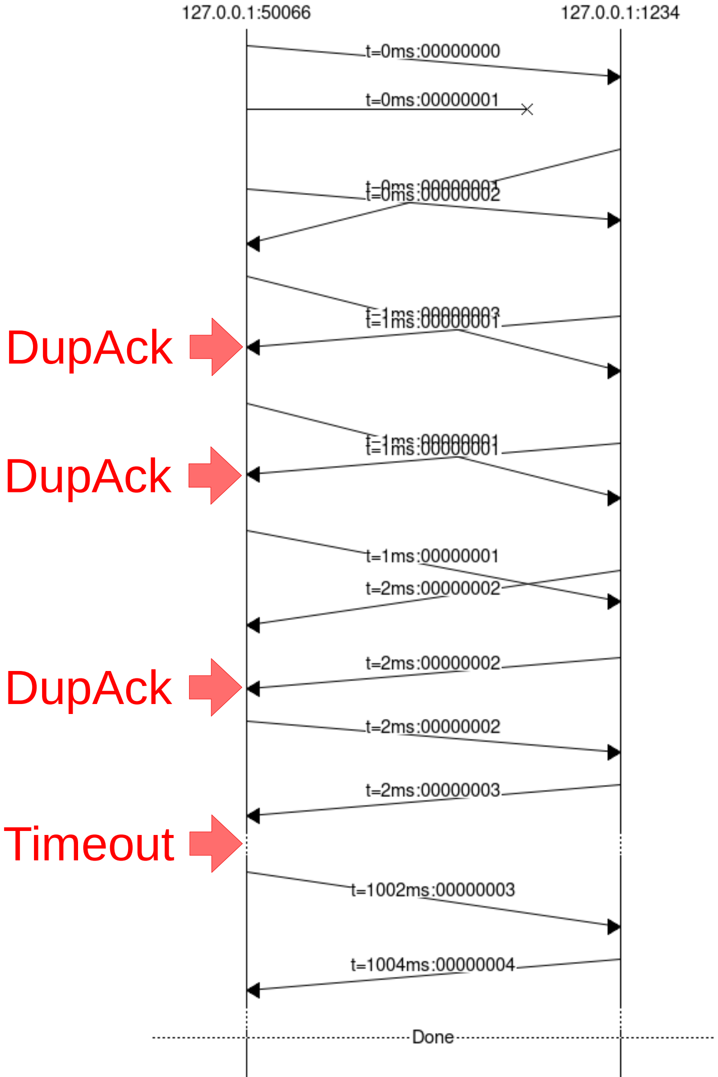
- 2 Chunks
- Sender
  - DROP_PATTERN="01"
  - Send Window = 1
- Receiver
  - DROP_PATTERN=""
  - Receive Window = 1



Advice: MSCs

Go-Back-N

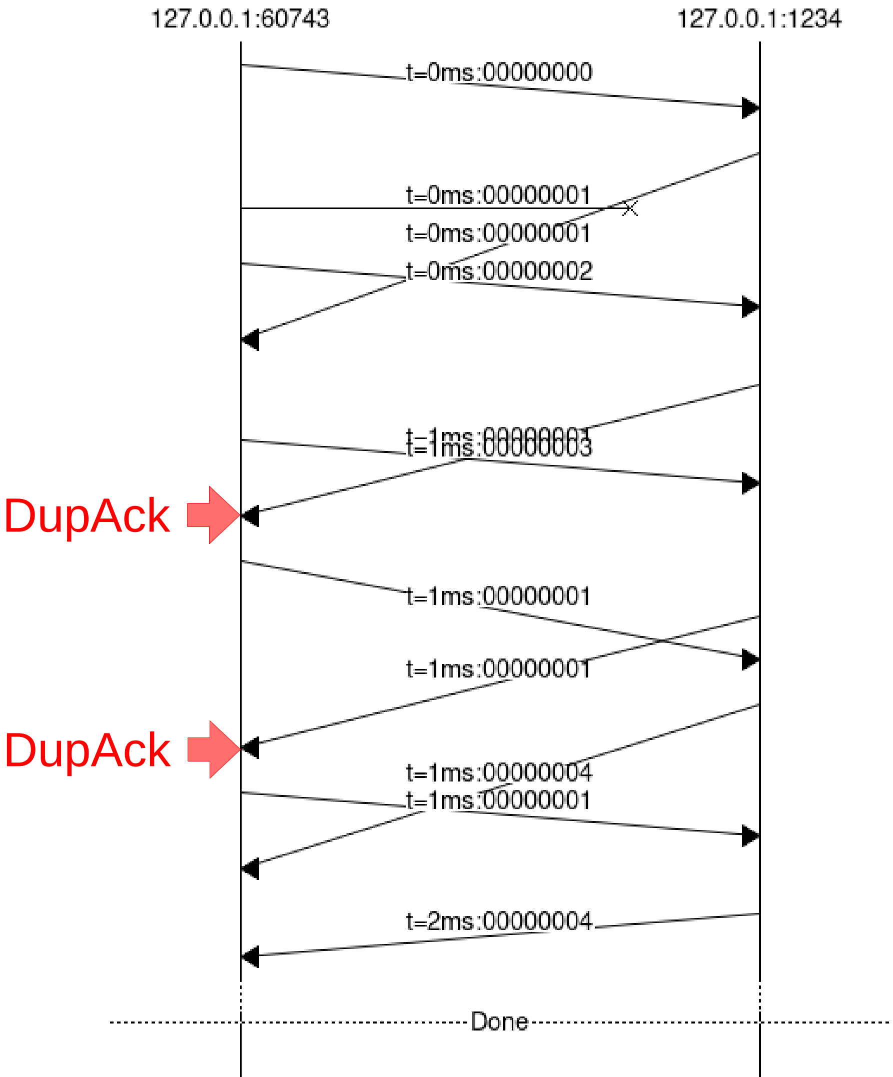
- 4 Chunks
- Sender
  - DROP_PATTERN="01"
  - Send Window = 3
- Receiver
  - DROP_PATTERN=""
  - Receive Window = 1



Advice: MSCs

Selective-Repeat

- 4 Chunks
- Sender
  - DROP_PATTERN="01"
  - Send Window = 3
- Receiver
  - DROP_PATTERN=""
  - Receive Window = 3



DupAck

DupAck

Advice: MSCs

Improv

- How Many Chunks?
- Sender
  - DROP_PATTERN="?"
  - Send Window = ?
- Receiver
  - DROP_PATTERN="?"
  - Receive Window = ?

?