Taking *Saratoga* from space-based ground sensors to ground-based space sensors

Lloyd Wood
Centre for Communication Systems Research, University of Surrey
l.wood@surrey.ac.uk

Charles Smith
Commonwealth Scientific and Industrial Research Organization (CSIRO)
charles.smith@csiro.au

Wesley M. Eddy
MTI Systems
wes@mti-systems.com

Will Ivancic
NASA Glenn Research Center
william.d.ivancic@grc.nasa.gov

Chris Jackson
Surrey Satellite Technology Ltd
C.Jackson@sstl.co.uk

with thanks to

IEEE Aerospace conference, Big Sky, Montana, March 2011.

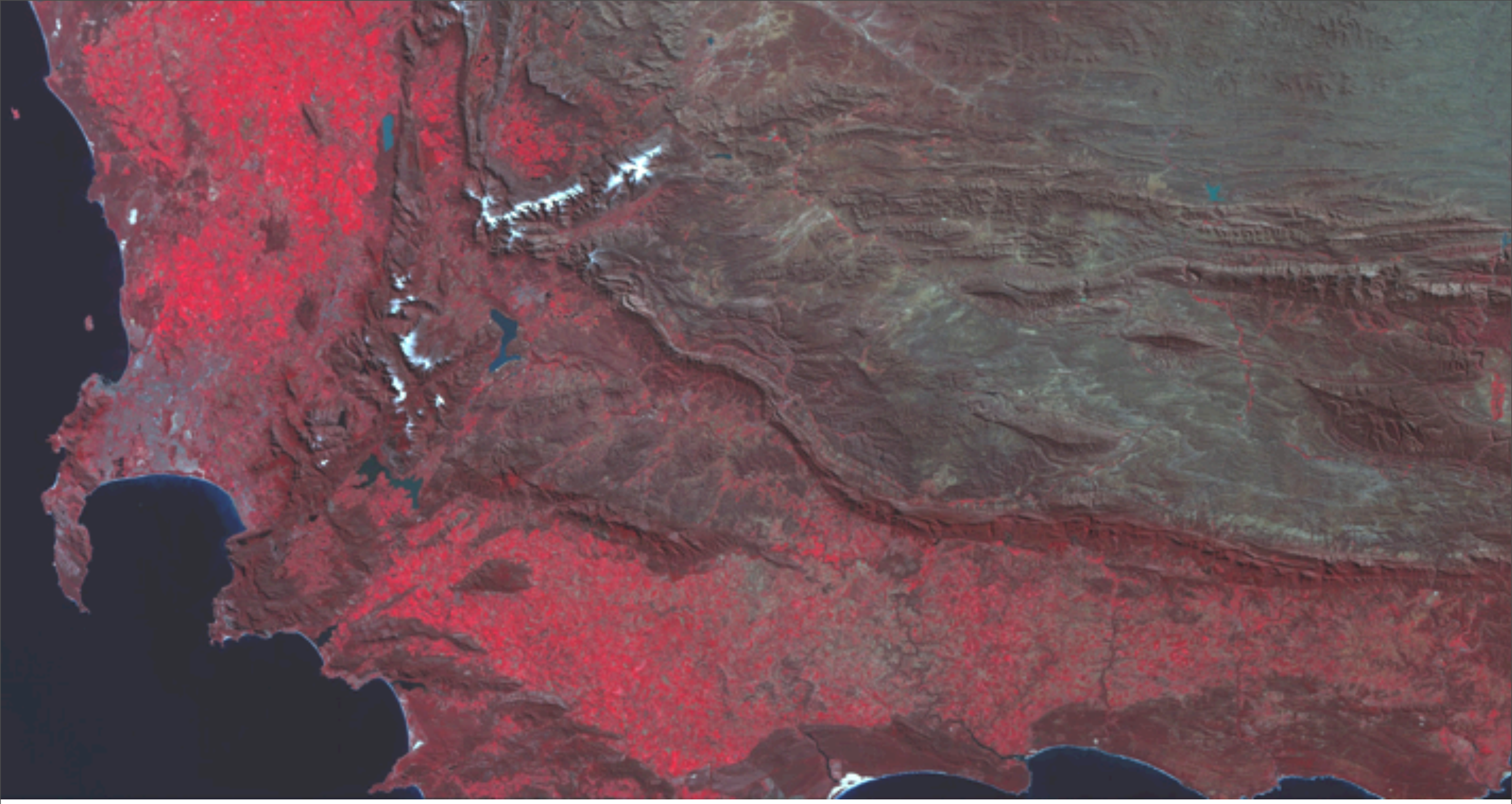
Saratoga - What's it good for?

Disaster Monitoring Constellation (DMC) satellites
built and operated by
Surrey Satellite Technology Limited (SSTL)

&

Challenges in Data Sensing, Transmission and Access
posed by New Radio Astronomy Telescopes
(Very Long Baseline Interferometers -
large distributed sensor nets)

Delivering sensor data!



Sensor data example: The Cape of Good Hope and False Bay.
False colours – red is vegetation.
Taken by UK-DMC satellite on the morning of Wednesday, 27 August 2008.
(Downloading this image also demonstrated optional “DTN bundle” use.)

DMC satellite characteristics

Let's use the UK-DMC satellite (first-generation DMC) as an example.

Onboard platform computer and CLEO Cisco router experimental payload

Three payload computers, the Solid-State Data Recorders (SSDRs):

- one with a StrongARM processor (legacy backup, GPS reflectometry)
- two with 200MHz Motorola MPC8260 PowerPC (run imaging cameras)
- RTEMS operating system (POSIX API, BSD sockets)
- Storage Capacity on PPC SSDRs: 1 GByte or 0.5 GByte RAM.
- **Embedded RTEMS OS code limit is 0.5 MByte - tiny!**

Asymmetry

- Uplink is S-Band at only 9600 bits per second
- Downlink is S-band at 8.134 Mbps (X-band systems now up to 210 Mbps)
- Datalink – Frame Relay/HDLC
- Network Protocol – IPv4 (could easily run IPv6)
- Transport Protocol (*Saratoga* version 0 over UDP)
  - *Saratoga* version 0 was developed by SSTL as a replacement for an implementation of CCSDS CFDP for simple, high-speed, low-overhead, file delivery from Low Earth Orbit to ground over highly asymmetric links.
  - UDP checksum turned off for speed as HDLC CRC provided sufficient reliability for SSTL's needs - and because traffic only goes one hop.

New Radio Sensor Types for Astronomy



Low-frequency dipole (70MHz - 450MHz)

H1 hydrogen line is at 1420.4MHz

Aperture Array (500MHz - 1GHz)

Phased-array feed (450MHz - 3GHz)



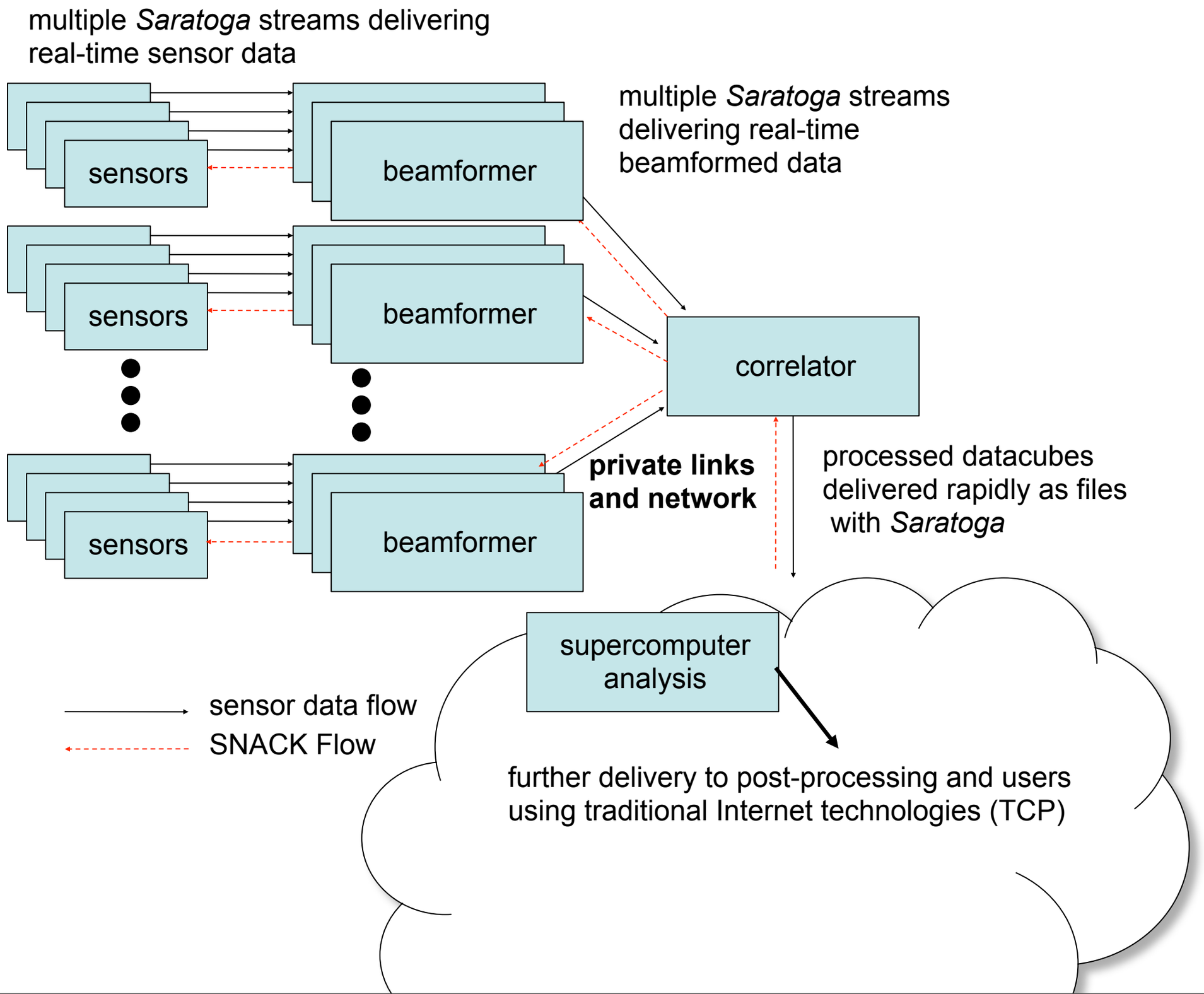
Australian Square Kilometre Array Pathfinder (ASKAP)

- A next-generation radio telescope being developed by the **Commonwealth Scientific and Industrial Research Organisation** (**CSIRO**) that incorporates novel receiver technologies and leading-edge Information Communication Technology systems.
- ASKAP will be one of the world's premier radio telescopes and will help to answer fundamental questions about our Universe. (And leads the way to the more powerful SKA.)
- 36 identical antennas, each 12 metres in diameter, working together as a single instrument.
  - Each dish holds 192 bi-polar phased-array feed sensors.
  - Each sensor generates a *10 Gbps* stream.
  - A total of 6,912 individual 10 Gbps streams – almost 70,000 Gbps, or **8.44 terabytes/second (TBps).**
  - This is transported over optical fiber - ideally reusing Ethernet and IP to leverage commercial technologies.

Murchison Radio-astronomy Observatory (MRO) is in the middle of *absolutely nowhere*.

(Deserts - clear skies, radio silence.)

Sensing and processing in an array



Data flow and processes in ASKAP

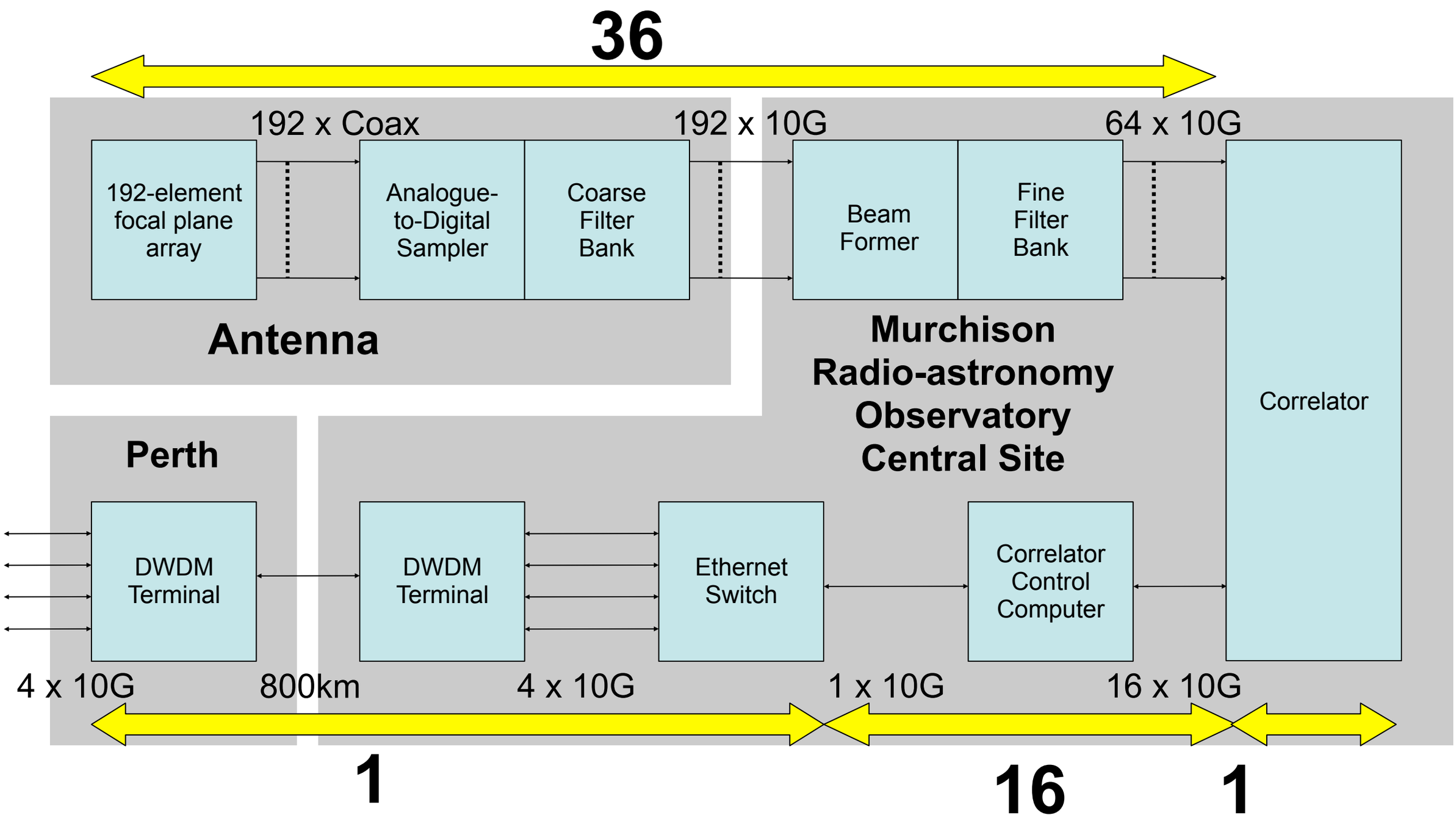
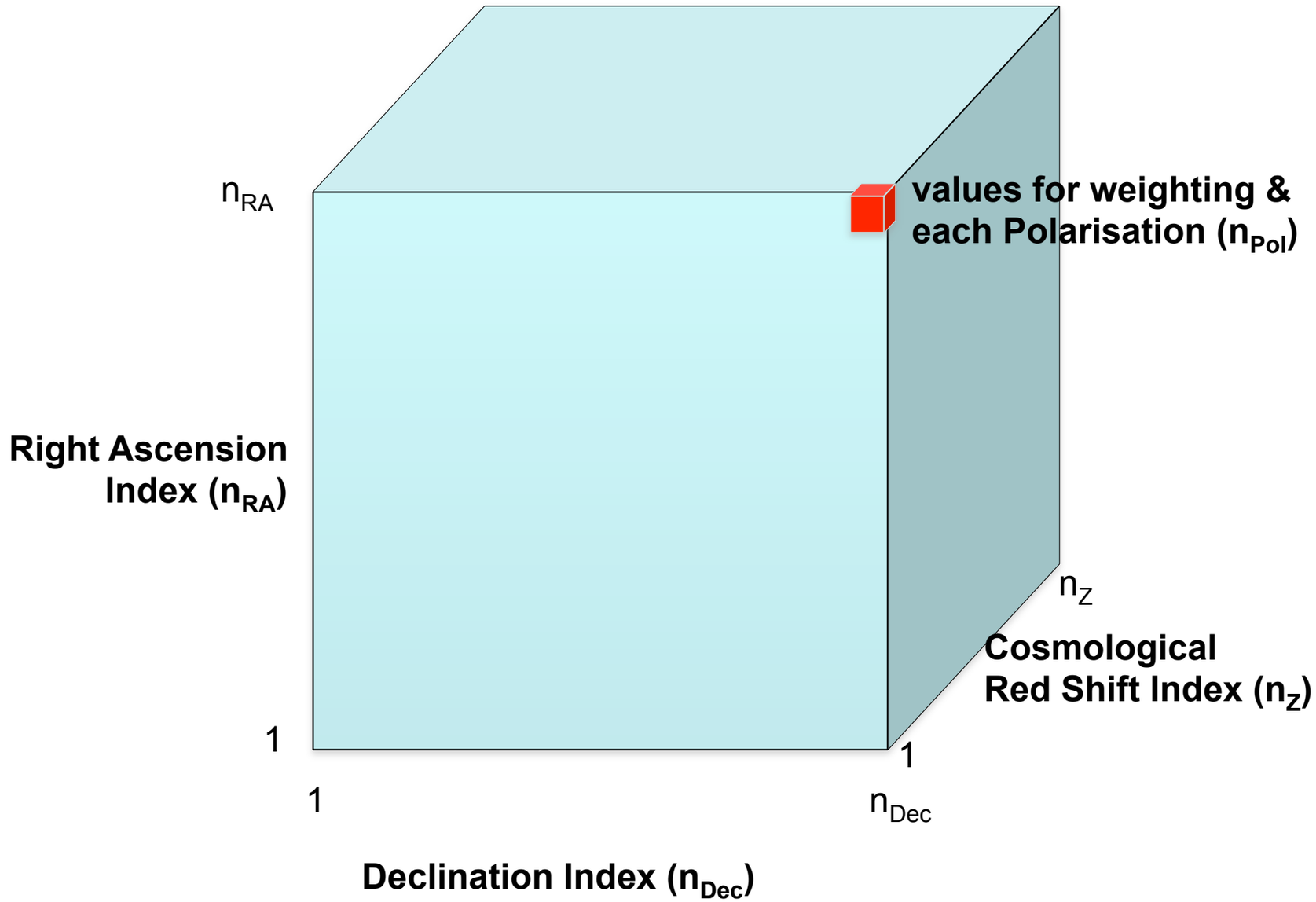


Image Data Cube



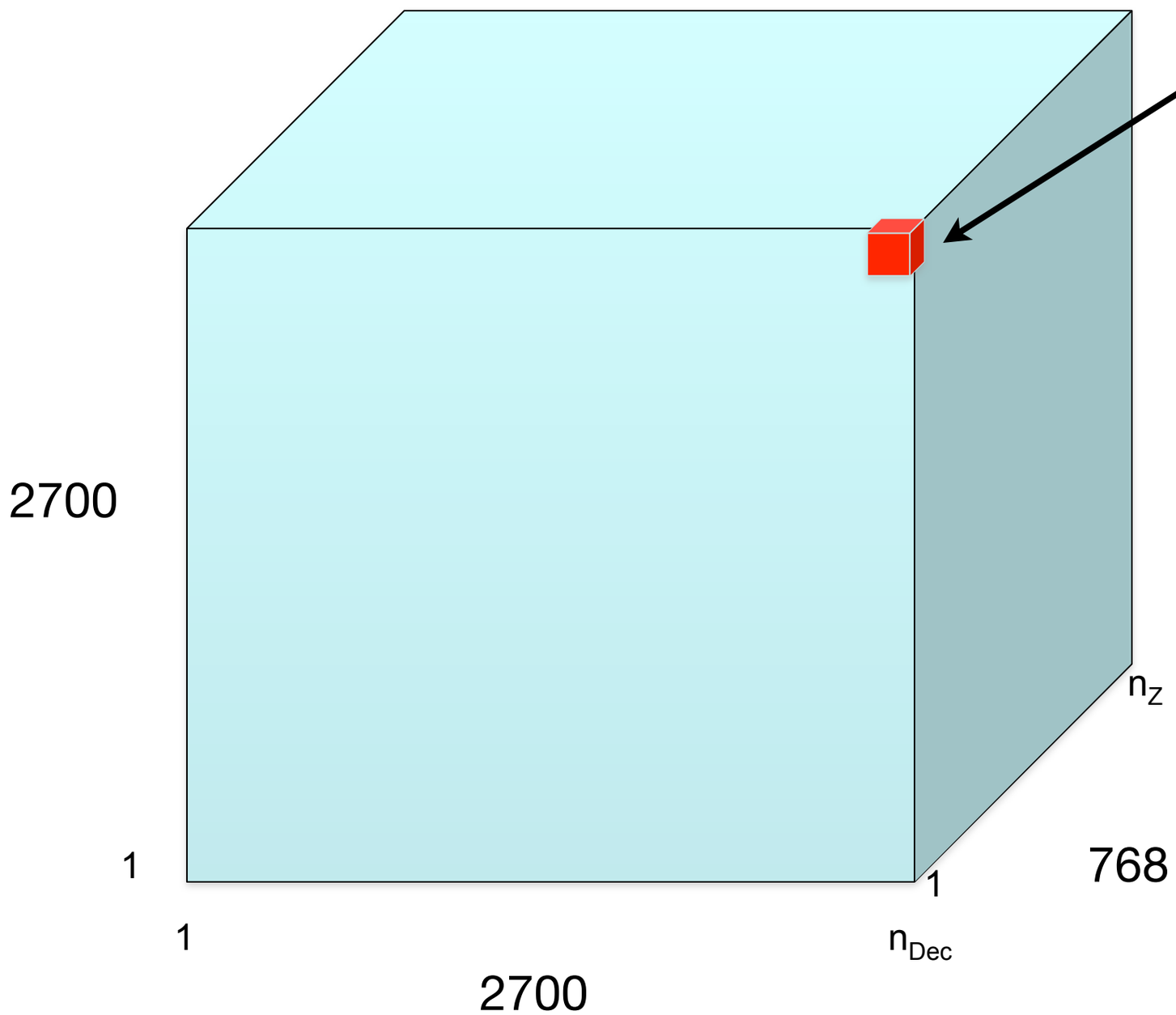
n_{pol} = number of polarization values per sample

fsize = size in single-precision floating point numbers (typically multiples of four bytes)

Data Image Cube Size (bytes) = $n_{Dec}\, n_{RA}\, n_Z n_{pol}$ *fsize*

Image Data Cubes for Murchison Widefield Array (MWA)

(Consists of 8,192 dual-polarization dipole antennas)



4 Polarizations
1 Weight
@ 4 bytes each

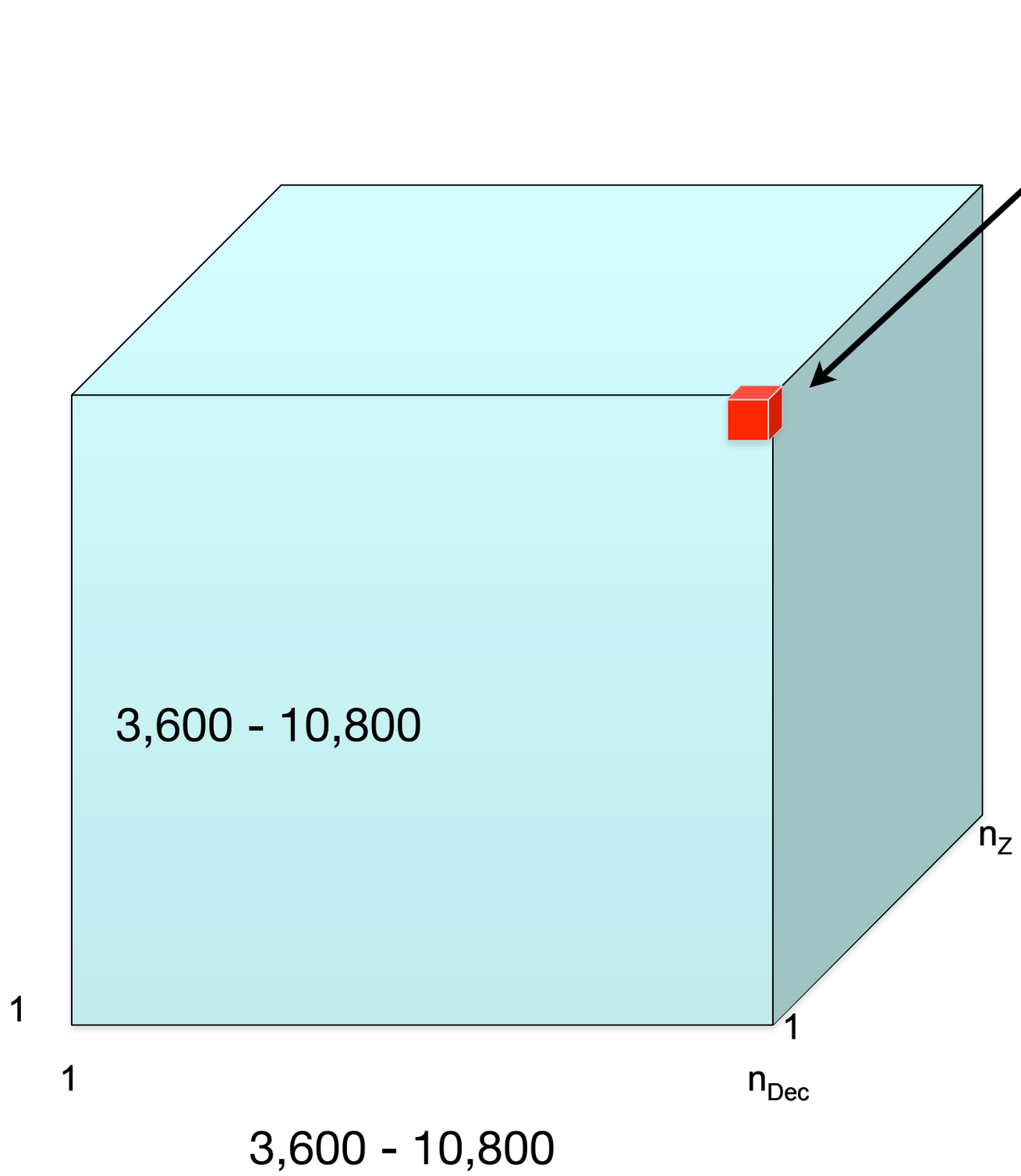
112 Gigabytes / Cube

One produced every 12 minutes

16 Terabytes / Day

5.9 Petabytes / Year

Image Data Cubes for ASKAP



53 Gigabytes
to 30 Terabytes/Cube

Wallaby all sky survey
1000 Cubes
Total 3.4 Petabytes

Dingo Deep Focus area
2 x 2500 hrs (50 cubes)
5 x 500 hrs (250 cubes)
Total 2.6 Petabytes

Image Data Cube transport

- Given a 10Gbps Ethernet Connection
  - 3.4-Terabyte image takes

    ~45 minutes to transport
  - 71-Terabyte visibility image takes

    ~15 hours to transport
- **And that's by filling the pipe completely and going flat out at line speed.**
- **A reliable, *high-speed* transport protocol is needed.**
- A single TCP-based transport flow just cannot fill this 10Gbps pipe. Don't want to wait around for TCP slow start and backoff to get up to speed.

Saratoga

A reliable, UDP-based,
file/stream/bundle transport protocol

What is *Saratoga* ?

- Version 0 developed by Surrey Satellite Technology Limited (SSTL) as a replacement to CFDP for simple high speed, low processing, file delivery from Low Earth Orbit to Ground over highly asymmetric links.
- *Saratoga* is a high-speed, UDP-based, peer-to-peer protocol, providing error-free guaranteed delivery of files, or streaming of data.
- Send data packets out as fast as you can. No specified congestion control is required, since data is usually only going one hop over a private link, or across high-speed, low-congestion private networks.
  - Some implementations have a rate-limiting option for restricted downstream links where line rate may not match downstream radio link
- No specified timers means no timeouts, so Saratoga is *ideal* for very long propagation delay networks (such as deep space).
- Every so often the transmitter asks for an acknowledgement from the file receiver. The receiver can also send acks if it thinks it needs to, or to start/restart/finish a transfer.
- Acks are Selective Negative Acknowledgements (SNACKs) indicating received packets, and any gaps to fill with resent data, including information so that intelligent sender rate control or congestion control can be provided if needed.
- Any multiplexing of flows is done by the *Saratoga* peers.
- *Saratoga* is an excellent protocol to use in asymmetric network topologies.

Saratoga is a reliable transport over UDP

Simple sliding window with selective acknowledgments.

- The HOLESTOFILL list on the receiver requests the transmitter to re-send frames that have not been properly received (a SNACK) by sending a STATUS with the list of HOLESTOFILL.
- The receive window only advances when offsets are contiguous. The left edge of the transmit window does not advance until the holes have been acknowledged by a HOLESTOFILL frame with an advanced offset.
- The UDP checksum is used per packet to cover both the header and payload. It is consistent, but not that strong (one's complement), and does not provide end-to-end guarantees for payloads sent using multiple packets.
- An optional end-to-end checksum, using one of CRC32/MD5/SHA-1, over the entire file being transferred, increases confidence that a reliable copy has been made, and that fragments have been reassembled correctly.

Optional features of *Saratoga* version 1

Specified to the IETF in an experimental internet-draft. Adds features.

Major features

- Scalable to handle large files. 16-bit descriptors for efficiency with small files. 128-bit descriptors cope with *huge* files up to 2^{128} bytes. 32- and 64-bit descriptors most useful.
- Streaming of data is supported. This allows *Saratoga* to be used for real-time delivery outside the file-based store-and-forward paradigm.

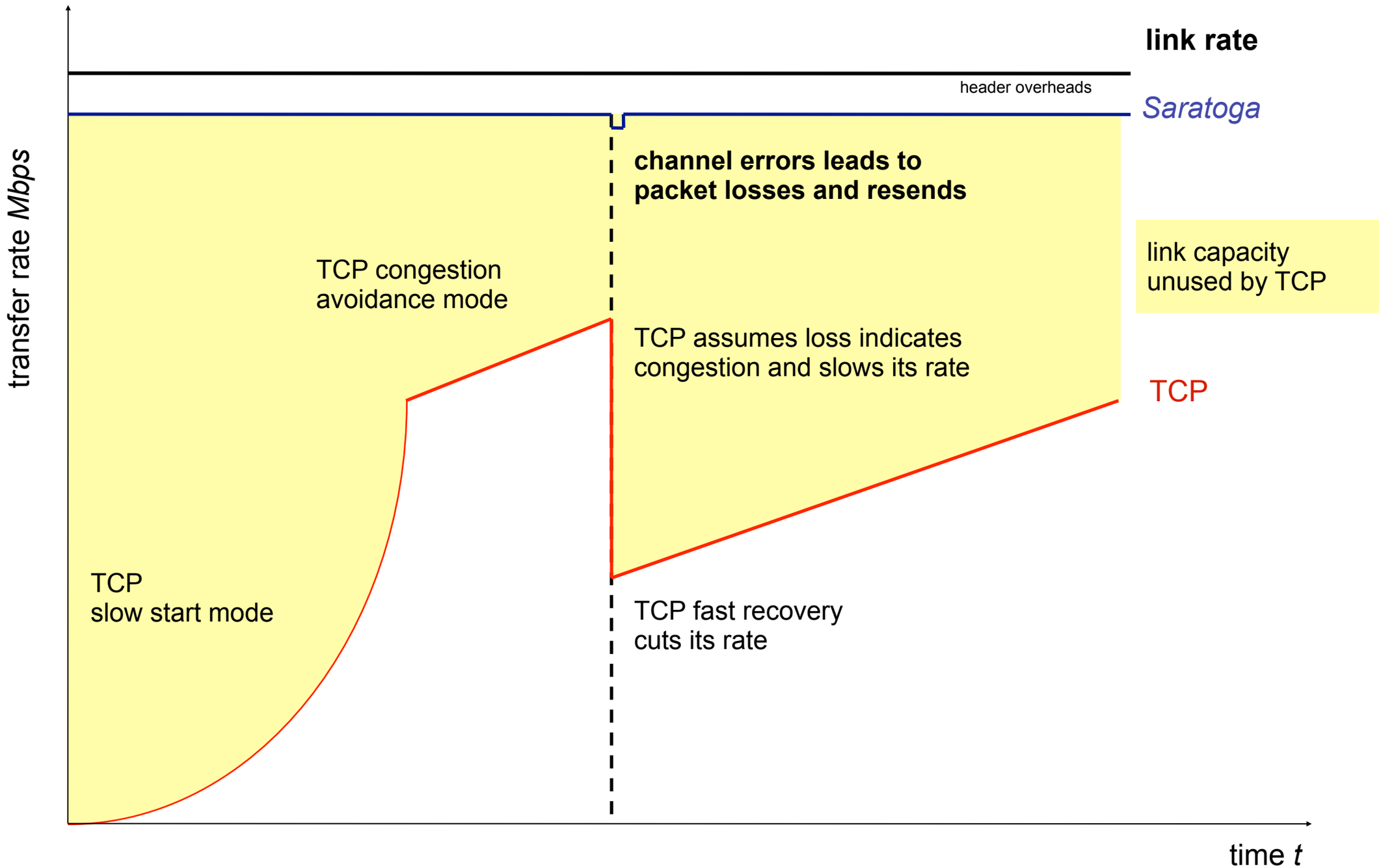
Minor features

- Supports link-local multicast to advertise presence, discover peers and for delivery to multiple receivers simultaneously for e.g. file or code image updates. (Will outperform TFTP trivial file transfer.)
- Optional UDP-Lite use for tolerating errors in payloads and minimizing checksum computation overhead. The UDP-Lite checksum covers a minimum of IP/UDP-Lite/*Saratoga* headers. The header content is always checked so that the information *about* the data is error-free.
- Optional “DTN bundle” delivery as a “bundle convergence layer”. Shown with tests from the UK-DMC satellite.

Why *Saratoga* instead of FTP/TCP ?

- For high throughput and link utilization on dedicated links, where a single TCP flow cannot fill the link to capacity.
- For links where TCP's assumptions about loss/congestion/ competition simply don't hold. i.e. High speed bulk transfer.
- There is no such thing as "slow-start" specified in *Saratoga*.
- Able to cope with high forward/back network asymmetry (>850:1).
- Long path-delay use – eventually TCP will fail to open a connection because its SYN/ACK exchange won't complete. TCP has many unwanted timers.
- Simplicity. TCP is really for a conversation between two hosts; needs a lot of code on top to make it transfer files. A focus on just moving files or streams of data makes sequence numbering simpler.
- Having SNACKs means that handling a sequence number wraparound when in streaming or bundling mode becomes easy.

Why *Saratoga* instead of FTP/TCP ?



What *Saratoga* does not do

- There is no MTU discovery mechanism, so you have to know the maximum packet size your network can transmit at. i.e. dictated by the frame size. This is okay for your own private network, but would be troublesome if used across the Internet.
- ***Saratoga* does not include "slow-start" or congestion control.** That is considered bad and unsociable behaviour on the Internet. *Saratoga* just blasts away on a link with no regard for other flows - which is the exact behaviour that makes it desirable in private networks and these environments!
- Simulations have shown that it is possible to implement congestion control mechanisms in *Saratoga* if desired - see our parallel University of Oklahoma paper describing Sender-Based TCP Friendly Rate Control.
  - Saratoga's timestamp option can be used to implement such *closed-loop* mechanisms.
  - Simple *open-loop* rate-limiting output to *XMbps* can also allow Saratoga to coexist with other traffic.

Saratoga Transactions

Saratoga Transactions

GET	Get a named file from the peer

Saratoga Transactions

GET	Get a named file from the peer
GETDIR	Get a directory listing of files from the peer

Saratoga Transactions

GET	Get a named file from the peer
GETDIR	Get a directory listing of files from the peer
DELETE	Delete a named file from the peer

Saratoga Transactions

GET	Get a named file from the peer
GETDIR	Get a directory listing of files from the peer
DELETE	Delete a named file from the peer
PUT	Put a named file or stream data to the peer

Saratoga Transactions

GET	Get a named file from the peer
GETDIR	Get a directory listing of files from the peer
DELETE	Delete a named file from the peer
PUT	Put a named file or stream data to the peer
PUTDIR	Put a directory listing of local files to the peer

Saratoga Frame Types

Saratoga Frame Types

BEACON

Sent periodically. Describes the *Saratoga* peer:

- Identity (e.g. EID)
- capability/desire to send/receive packets.
- max. file descriptor handled: 16/32/64/128-bit.

Saratoga Frame Types

BEACON	Sent periodically. Describes the *Saratoga* peer: Identity (e.g. EID) capability/desire to send/receive packets. max. file descriptor handled: 16/32/64/128-bit.
REQUEST	Asks for a file initiating ‘get’ transaction get file get directory listing delete a file.

Saratoga Frame Types

BEACON

Sent periodically. Describes the *Saratoga* peer:
- Identity (e.g. EID)
- capability/desire to send/receive packets.
- max. file descriptor handled: 16/32/64/128-bit.

REQUEST

Asks for a file initiating ‘get’ transaction
- get file
- get directory listing
- delete a file.

METADATA

Sent at start of transaction. Initiates a ‘put’ transaction.
- Describes the file, bundle or stream:
  - set identity for transaction
  - file name/details, including size.
  - set descriptor size offsets to be used for this transaction
    (16/32/64/128-bit pointer sizes.)

Saratoga Frame Types

BEACON

Sent periodically. Describes the *Saratoga* peer:
- Identity (e.g. EID)
- capability/desire to send/receive packets.
- max. file descriptor handled: 16/32/64/128-bit.

REQUEST

Asks for a file initiating 'get' transaction
- get file
- get directory listing
- delete a file.

METADATA

Sent at start of transaction. Initiates a 'put' transaction.
- Describes the file, bundle or stream:
  - set identity for transaction
  - file name/details, including size.
  - set descriptor size offsets to be used for this transaction
    - (16/32/64/128-bit pointer sizes.)

DATA

Actual Data.
- Uses descriptor of chosen size to indicate offset for data segment in the file/bundle or stream.

Saratoga Frame Types

BEACON

Sent periodically. Describes the *Saratoga* peer:
- Identity (e.g. EID)
- capability/desire to send/receive packets.
- max. file descriptor handled: 16/32/64/128-bit.

REQUEST

Asks for a file initiating ‘get’ transaction
- get file
- get directory listing
- delete a file.

METADATA

Sent at start of transaction. Initiates a ‘put’ transaction.
- Describes the file, bundle or stream:
  - set identity for transaction
  - file name/details, including size.
  - set descriptor size offsets to be used for this transaction
    (16/32/64/128-bit pointer sizes.)

DATA

Actual Data.
- Uses descriptor of chosen size to indicate offset for data segment in the file/bundle or stream.

STATUS

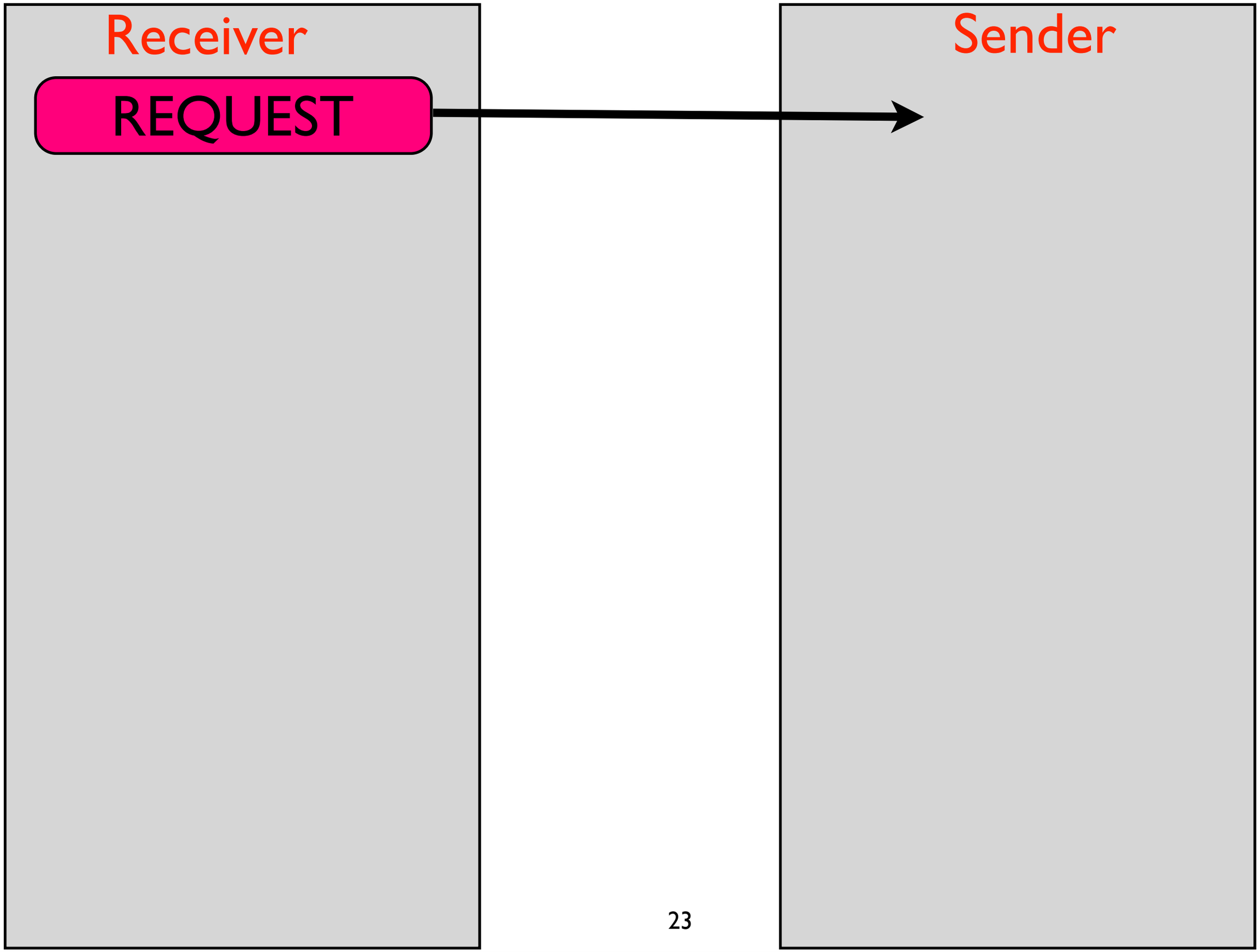
Missing Data Offsets / Error & Status Messages
- Selective negative ack (‘snack’) HOLESTOFILL data.
- Set left window edge for successful transfer so far
- List of offsets and lengths indicate missing ‘holes’ in data.

Transaction GET or GETDIR

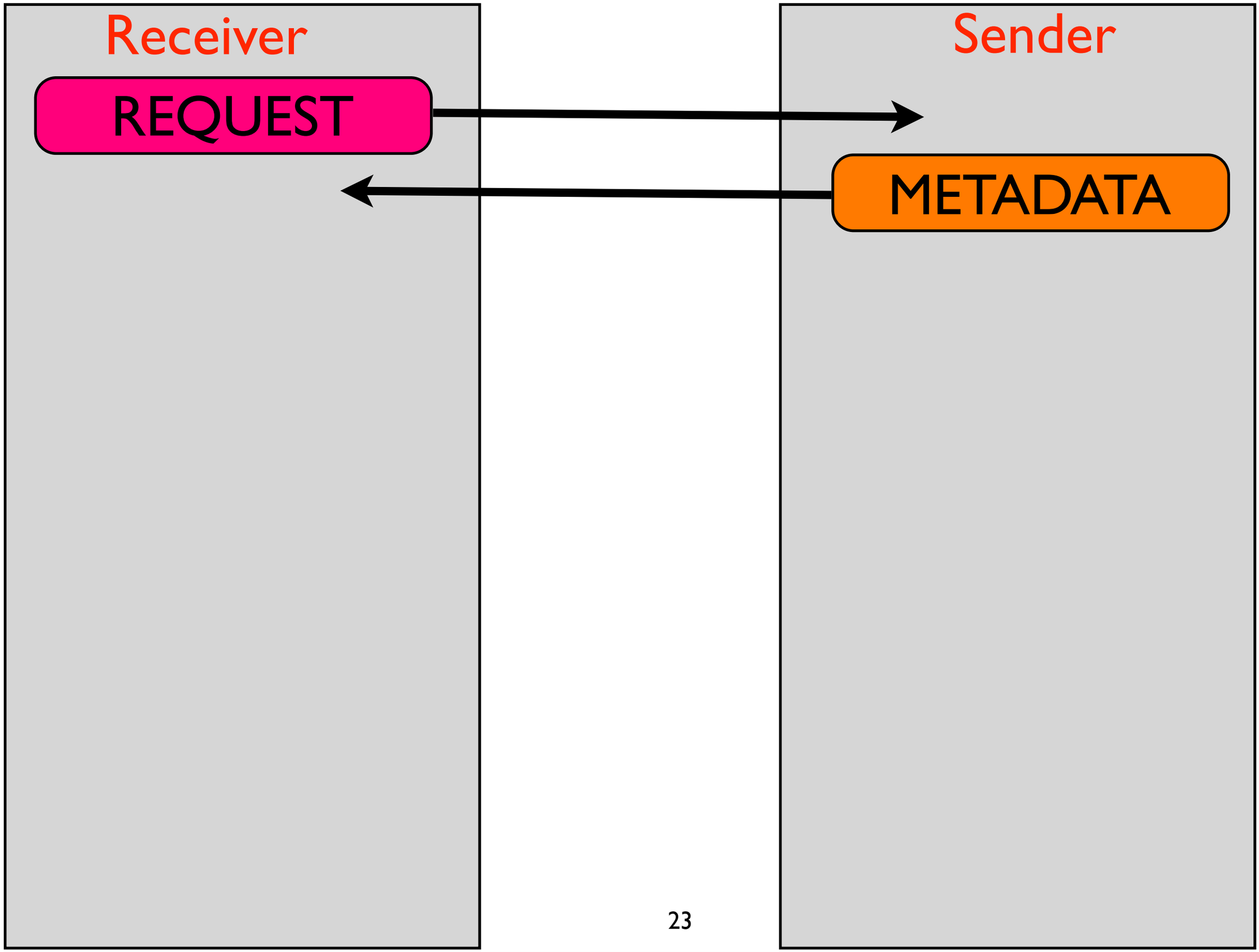
Receiver

Sender

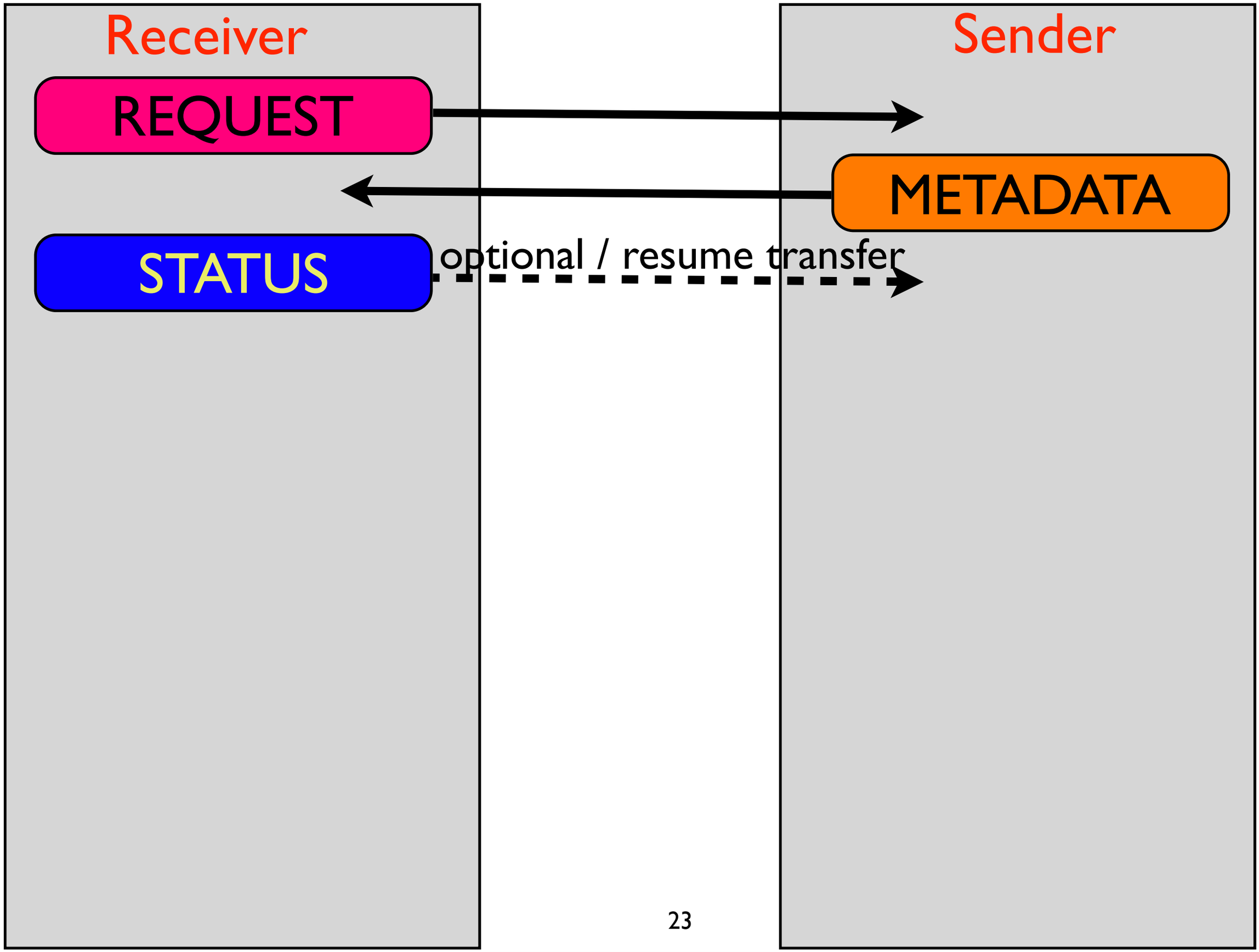
Transaction GET or GETDIR



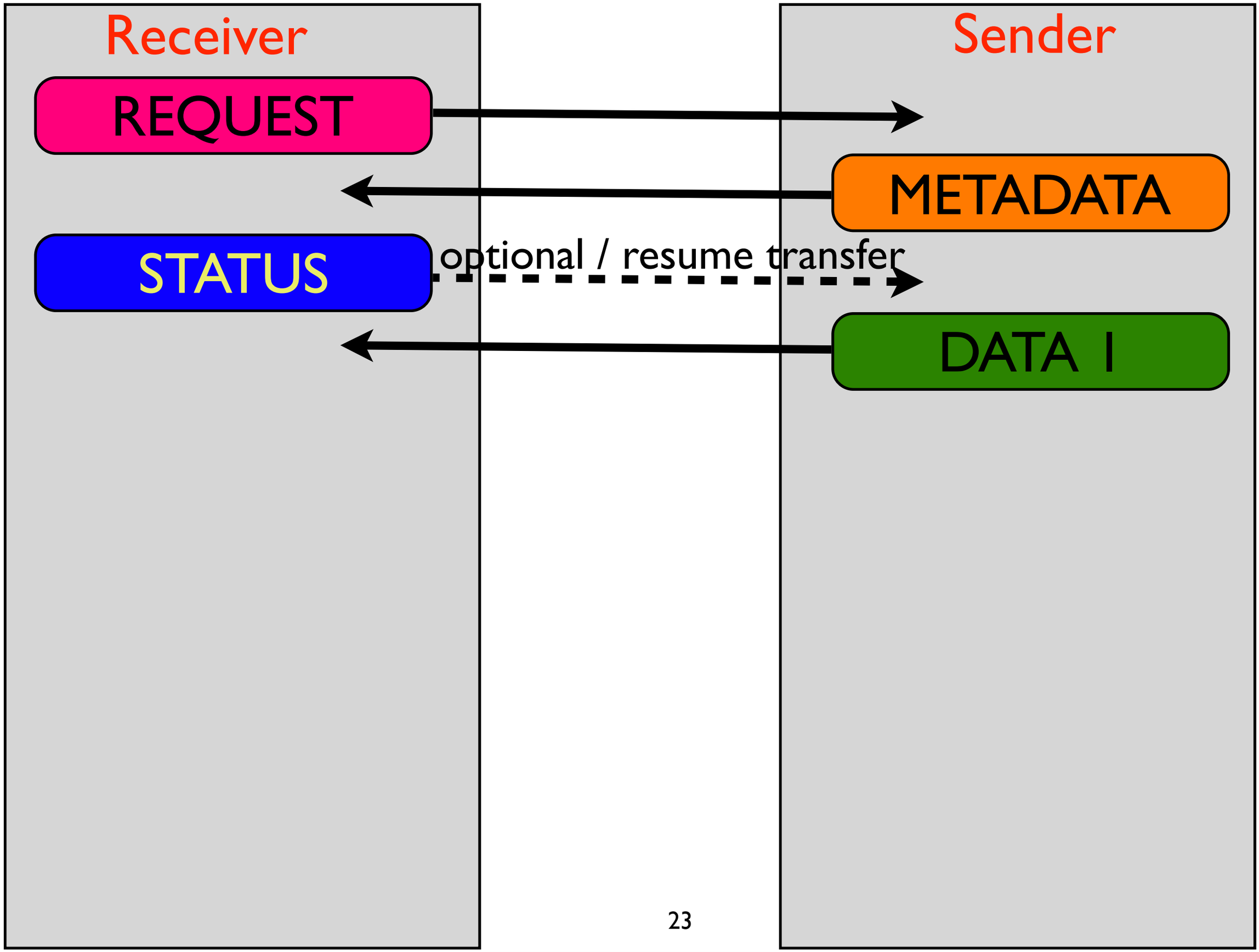
Transaction GET or GETDIR



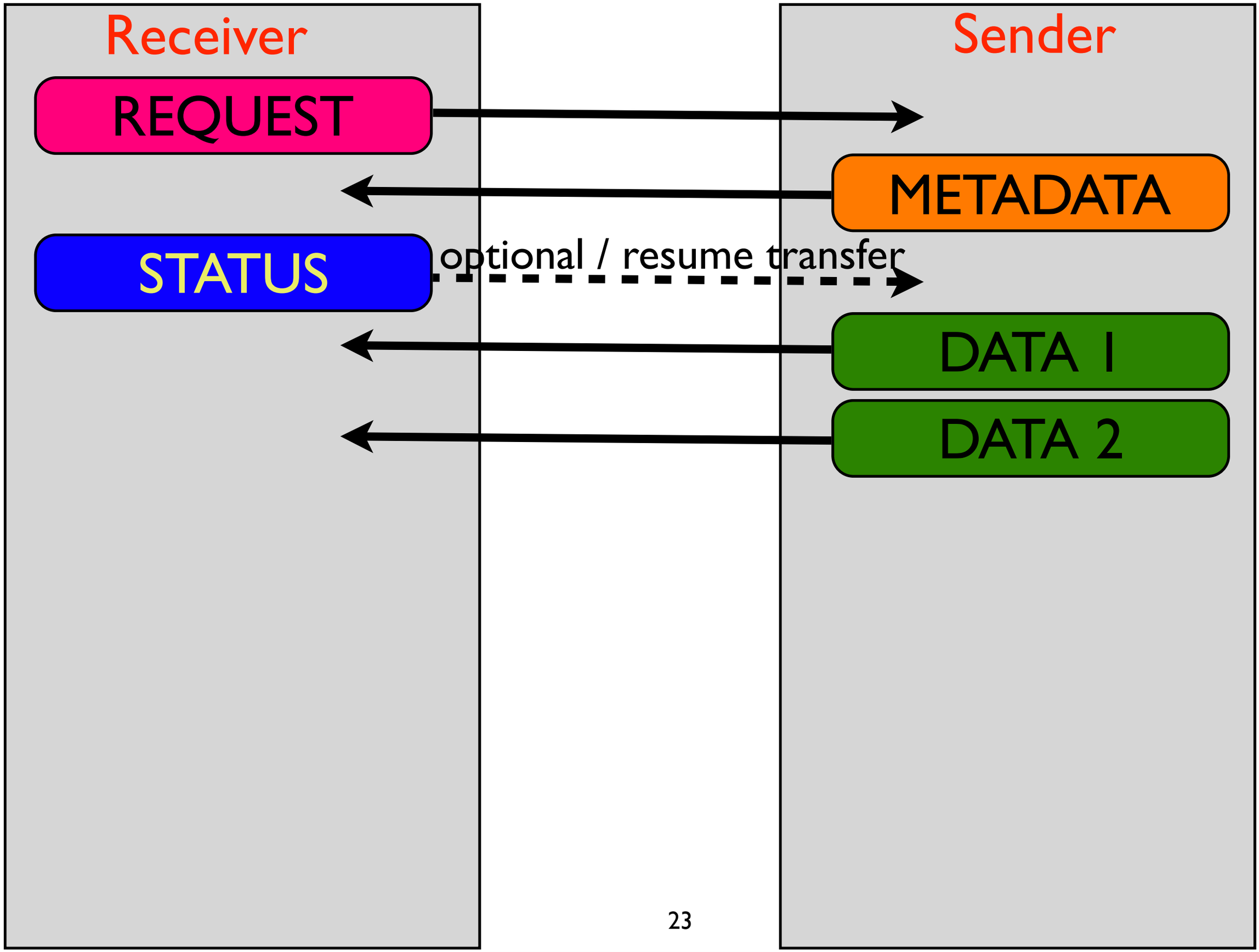
Transaction GET or GETDIR



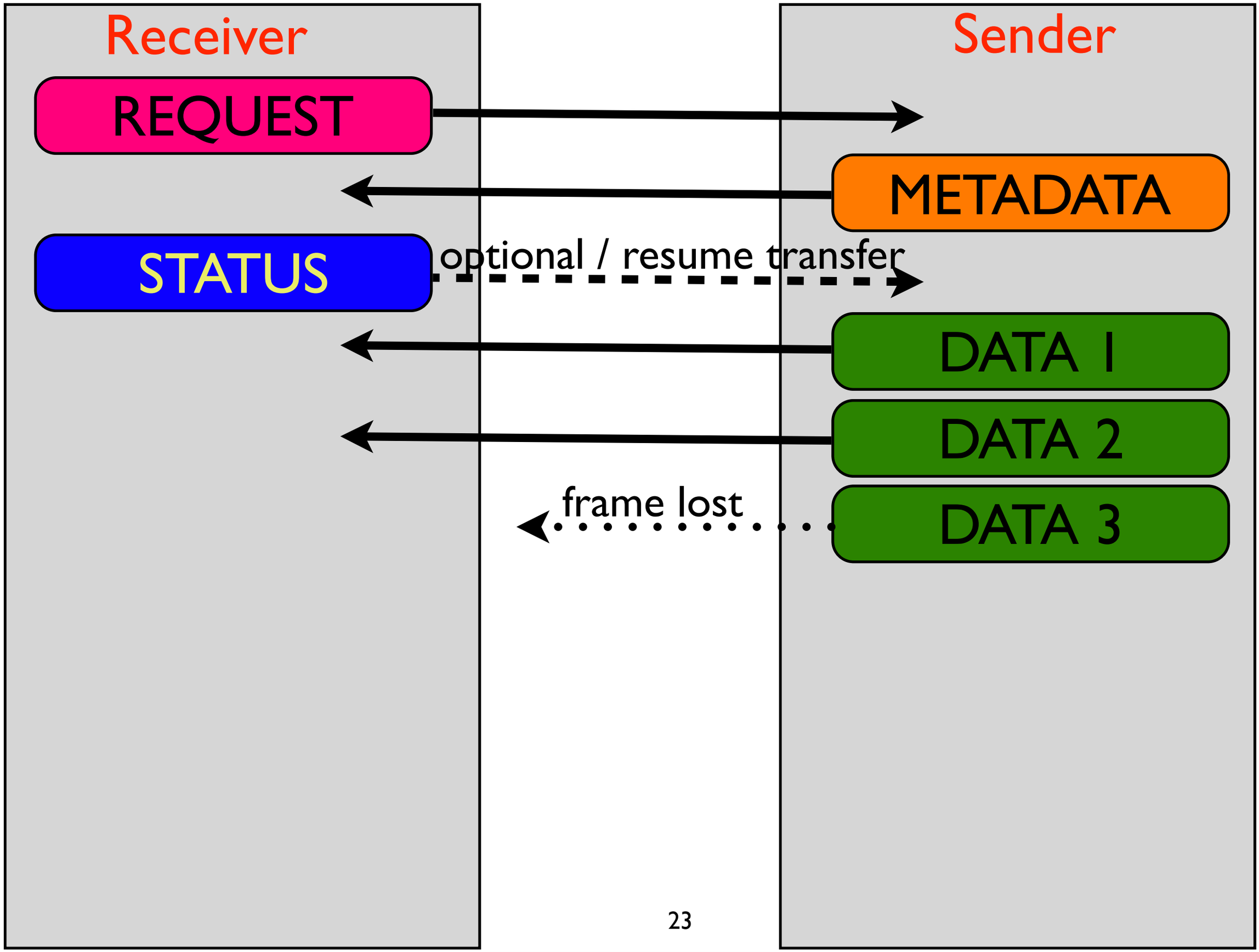
Transaction GET or GETDIR



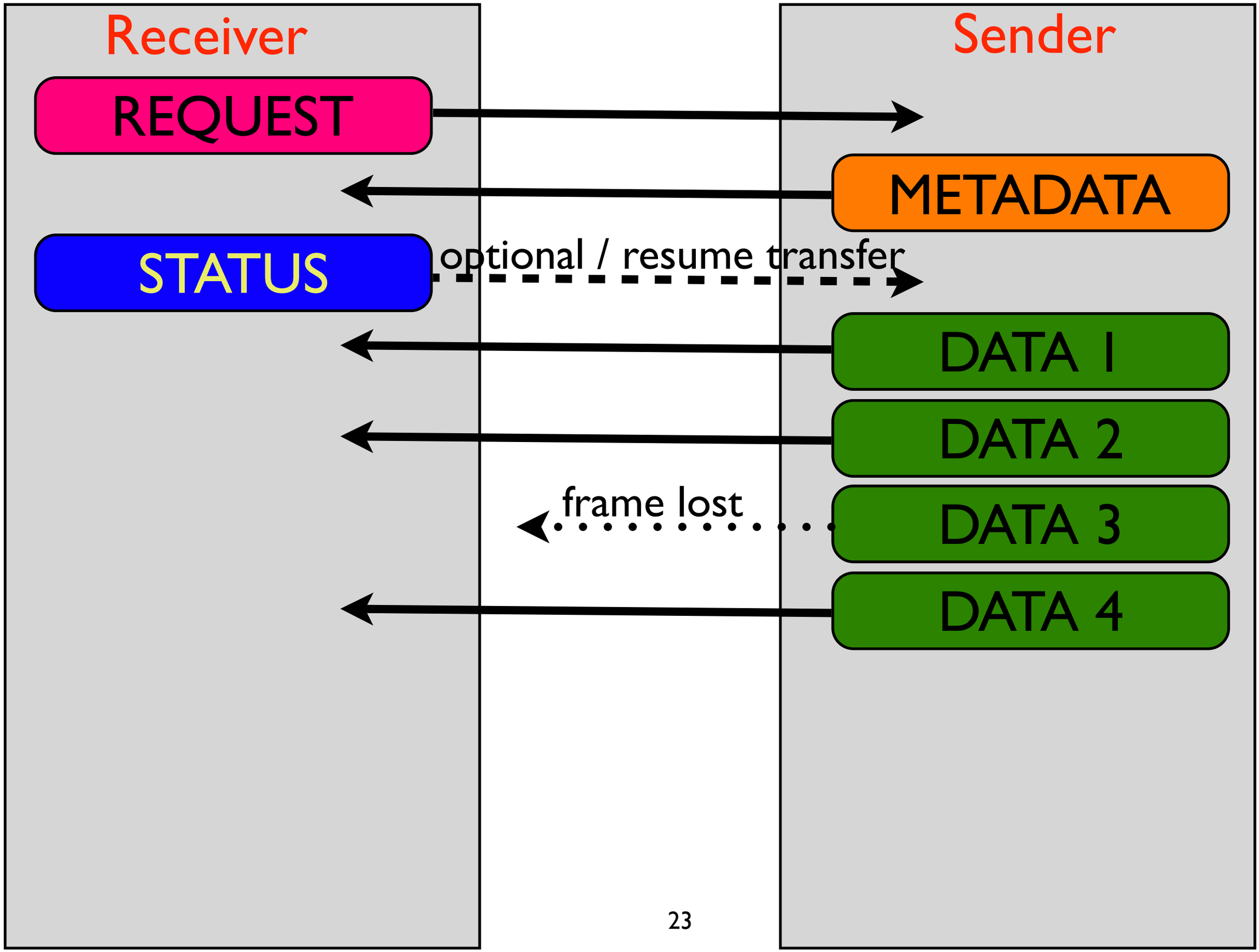
Transaction GET or GETDIR



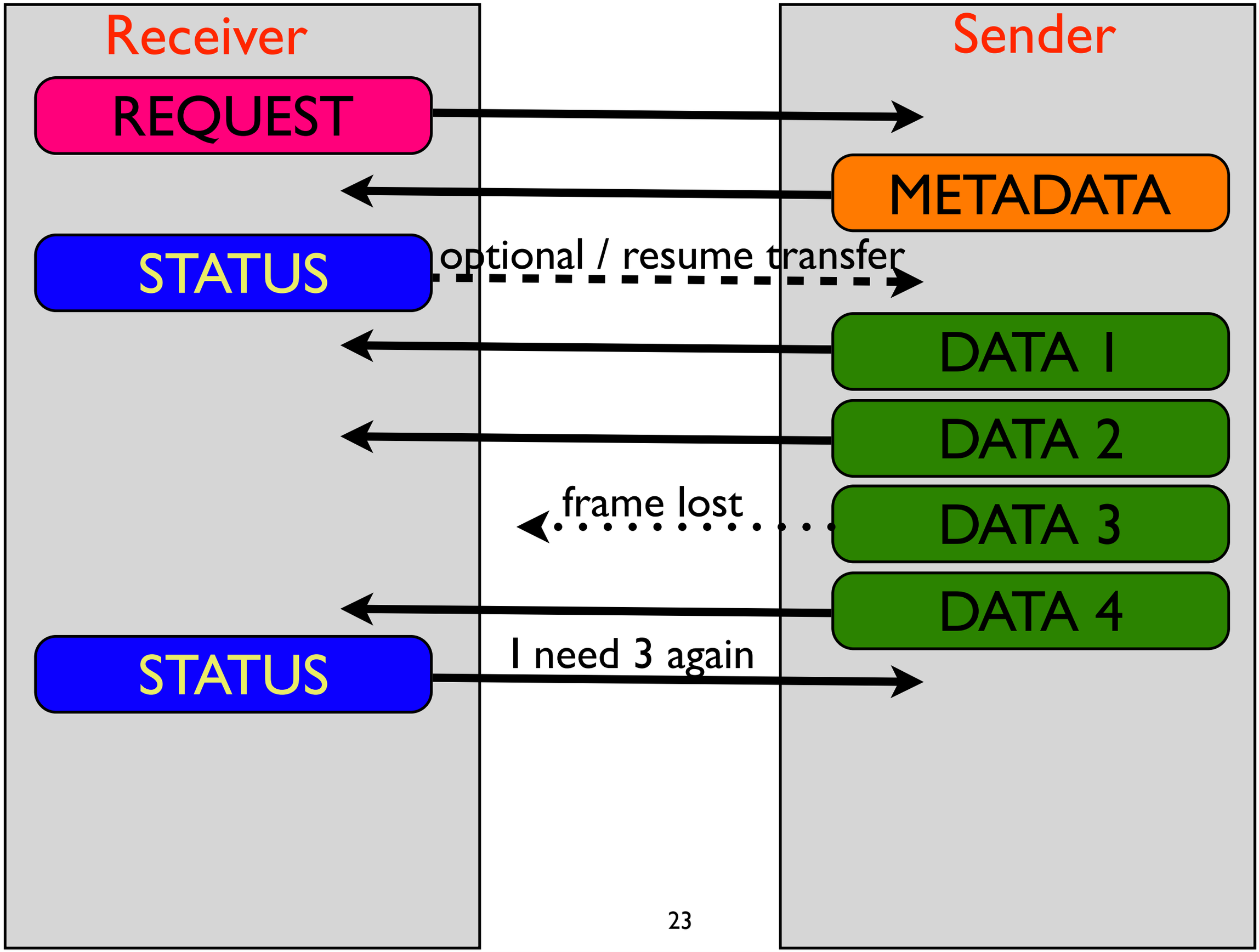
Transaction GET or GETDIR



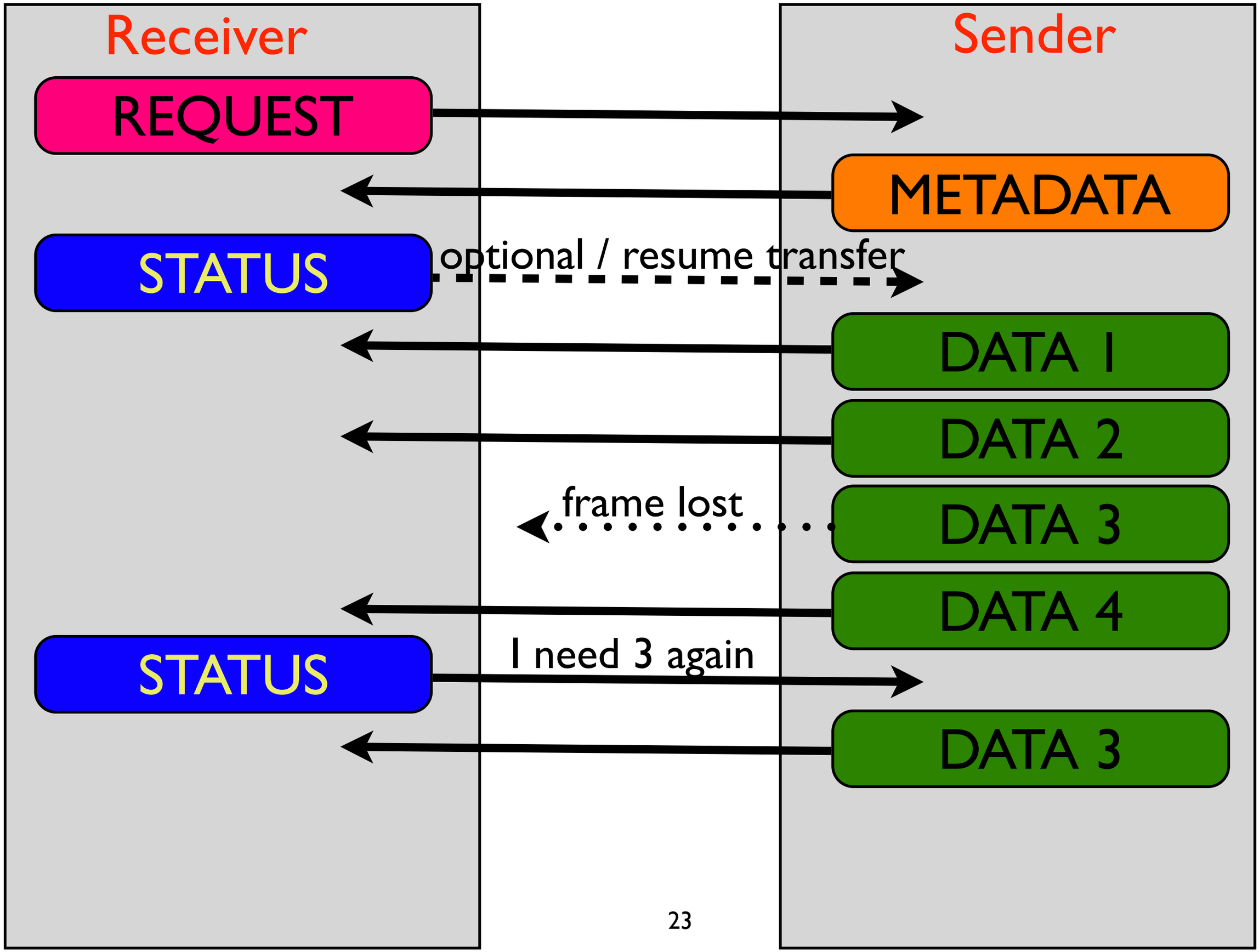
Transaction GET or GETDIR



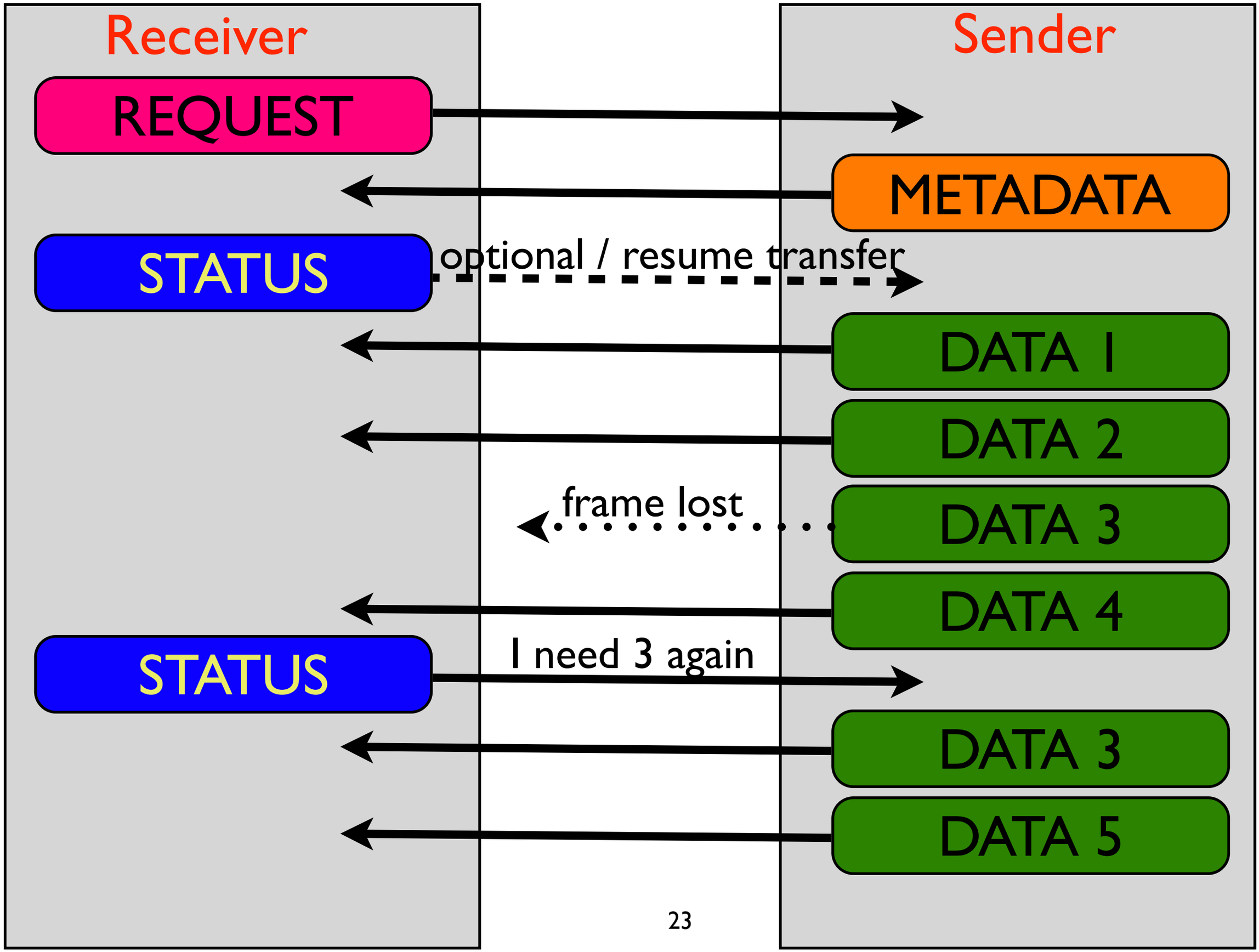
Transaction GET or GETDIR



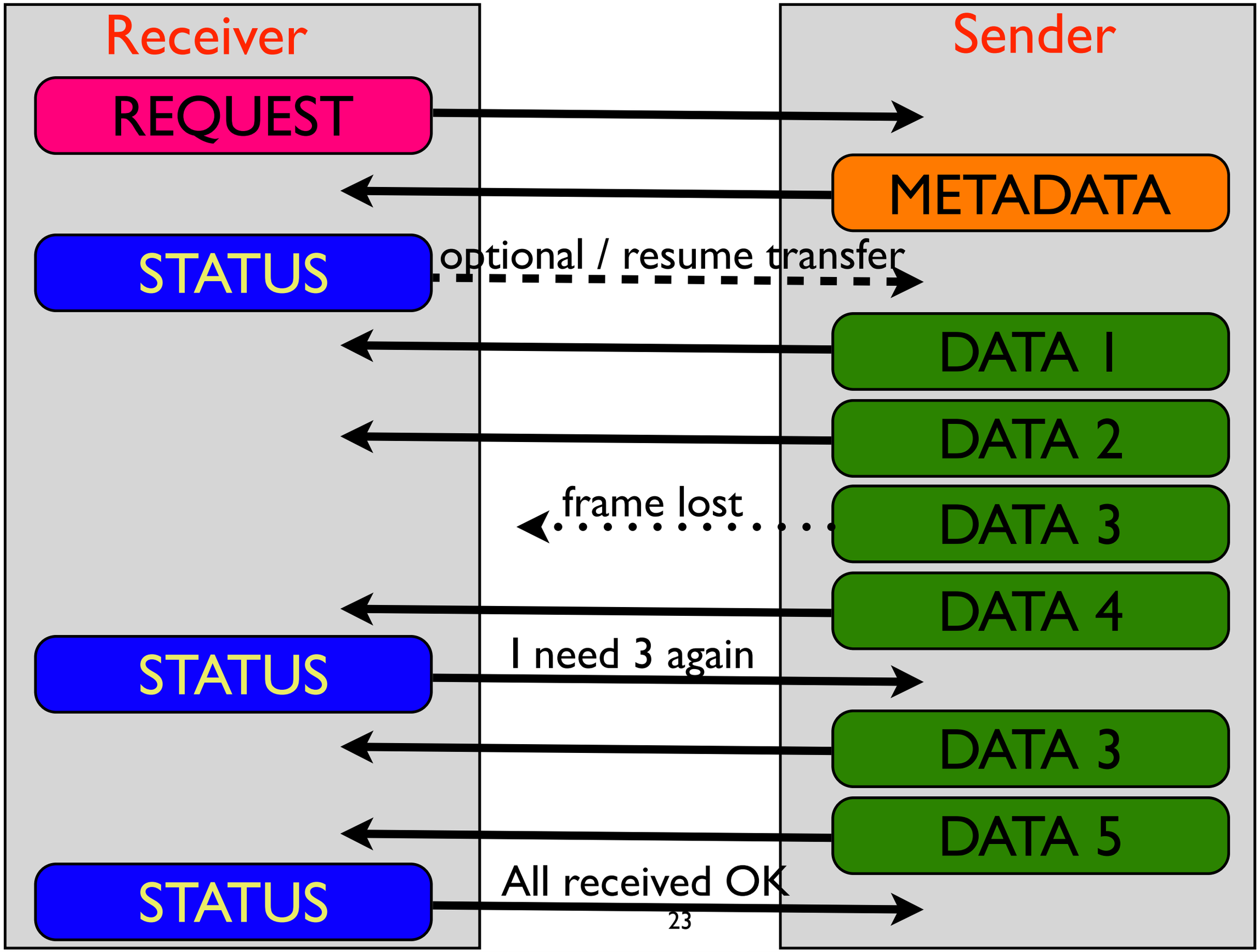
Transaction GET or GETDIR



Transaction GET or GETDIR



Transaction GET or GETDIR

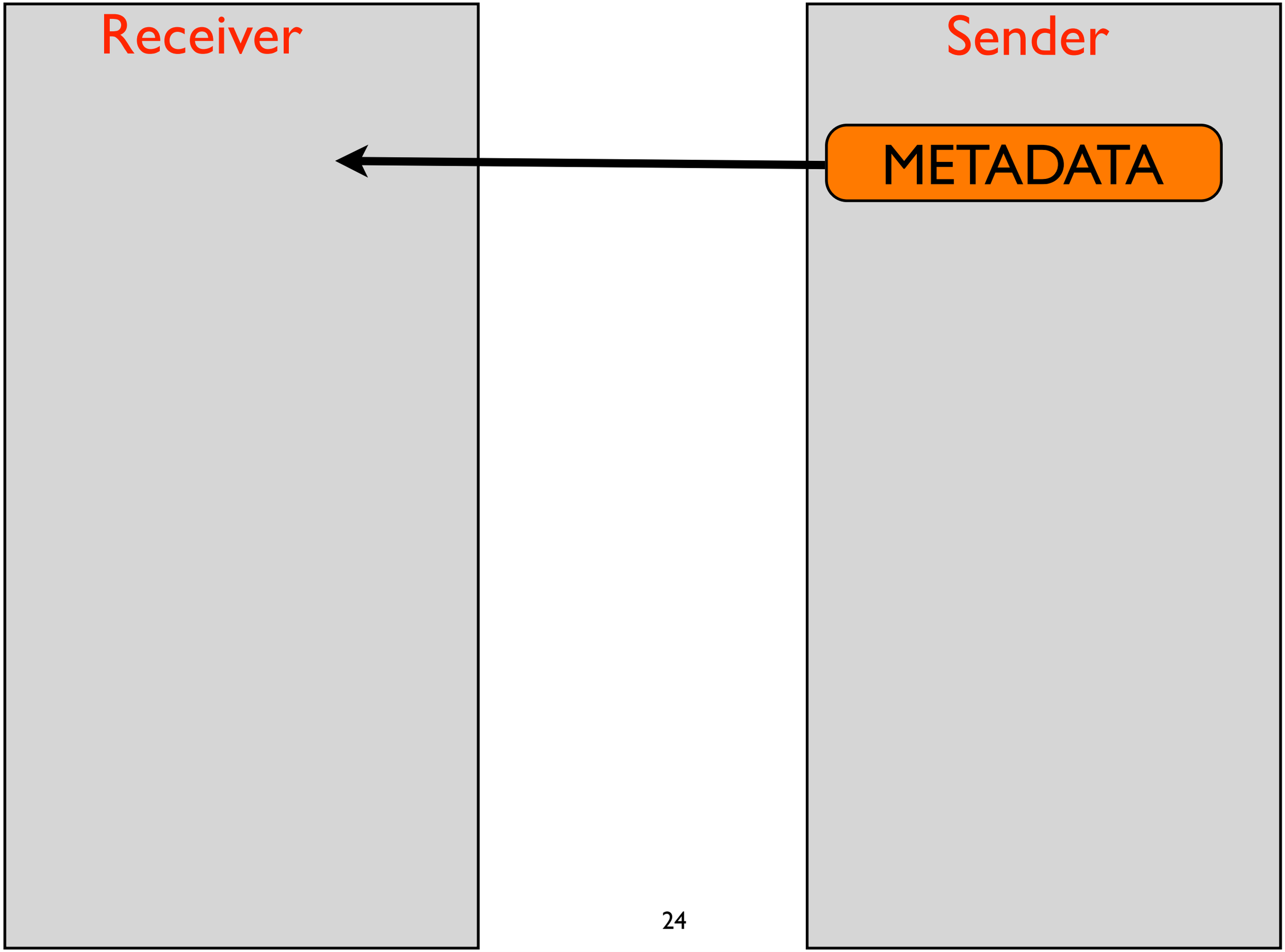


Transaction PUT or PUTDIR

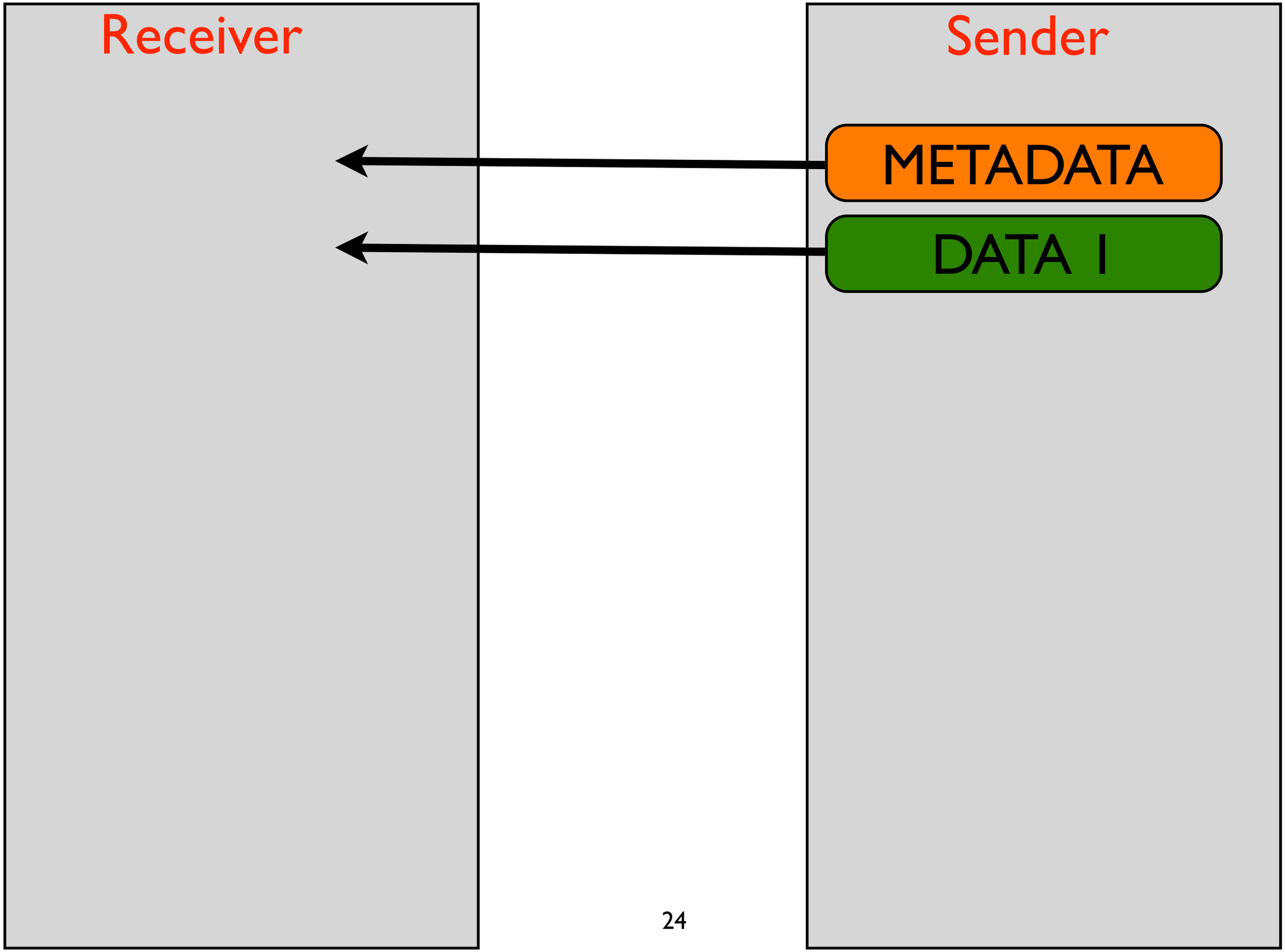
Receiver

Sender

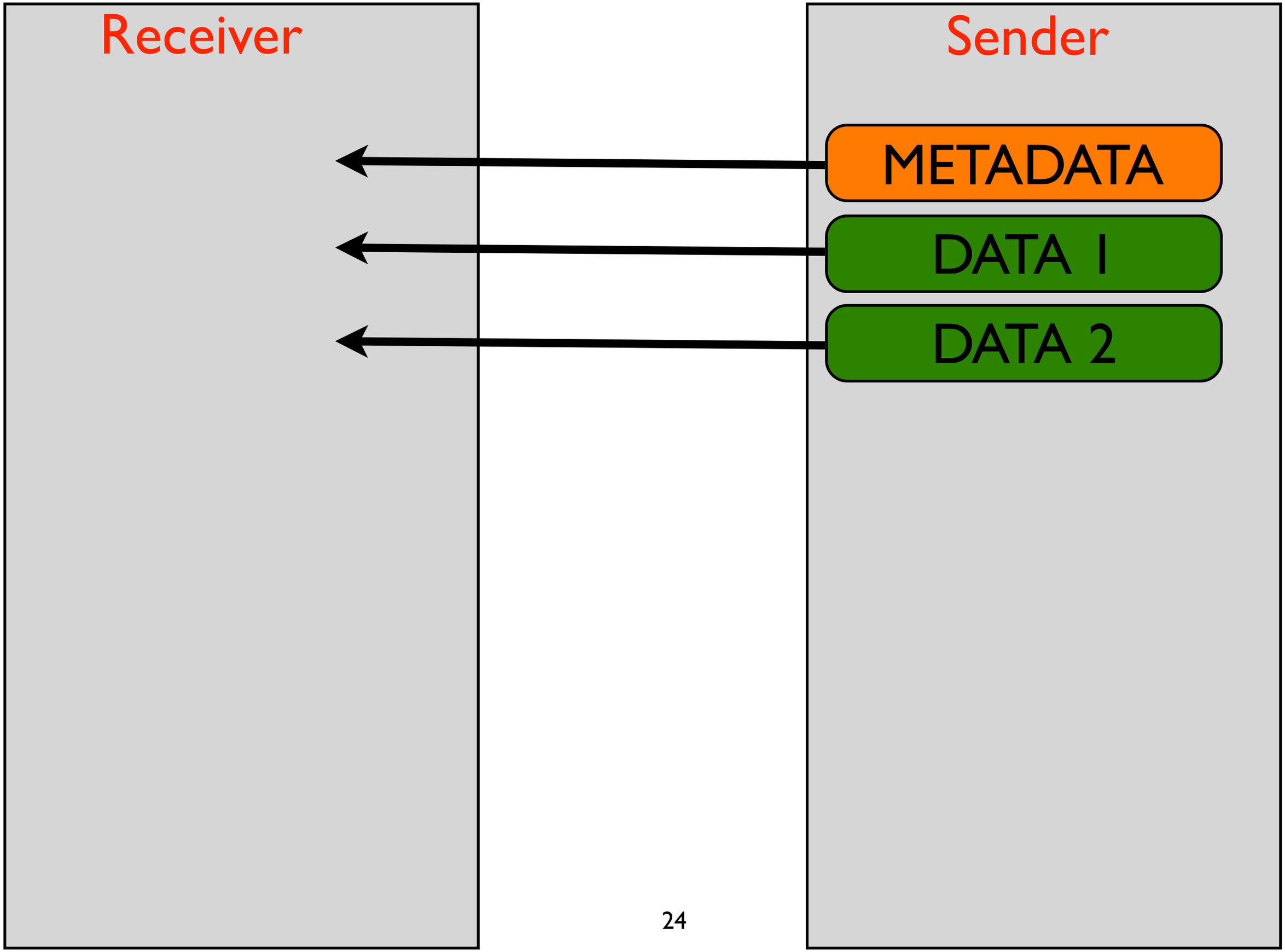
Transaction PUT or PUTDIR



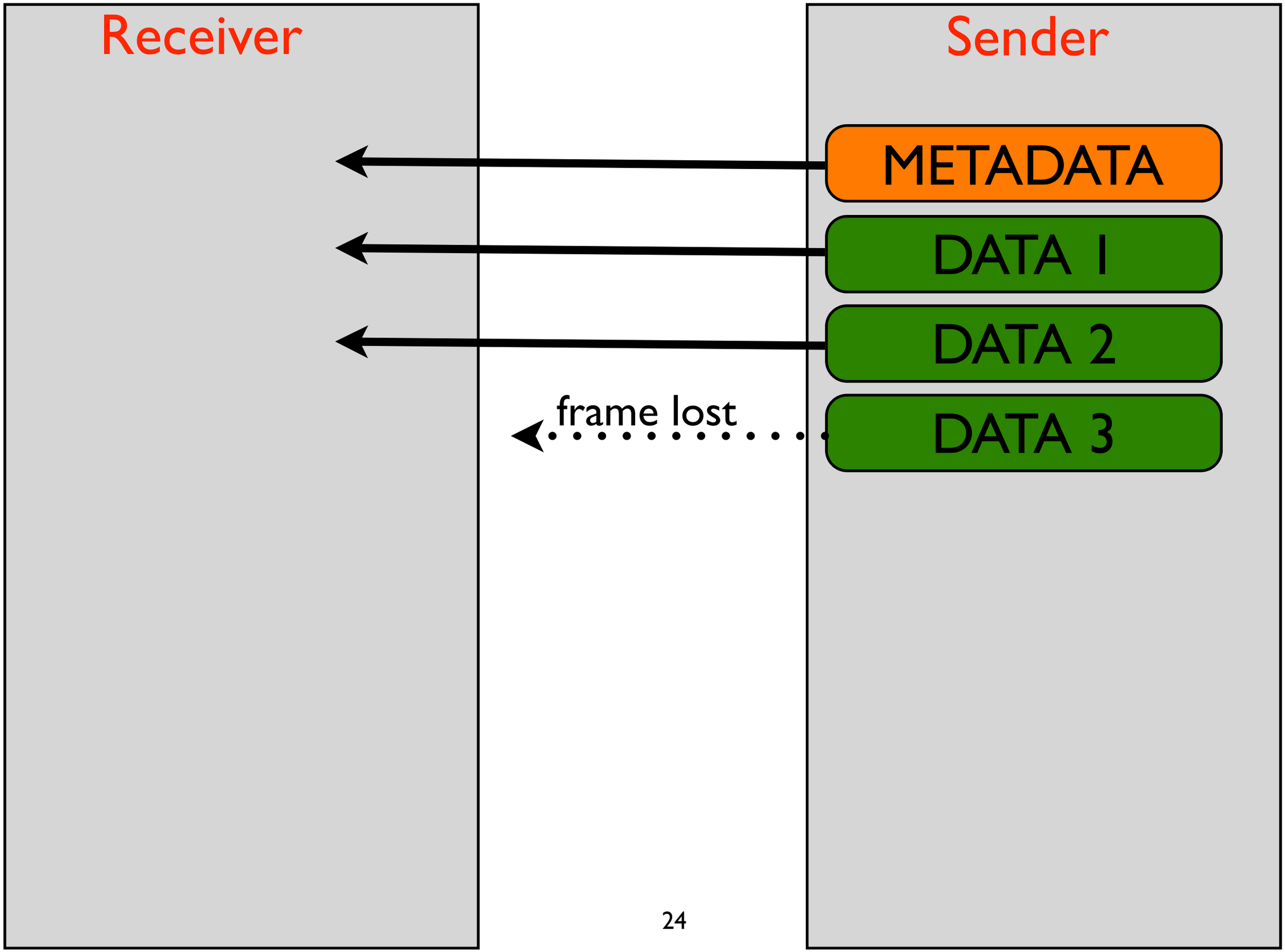
Transaction PUT or PUTDIR



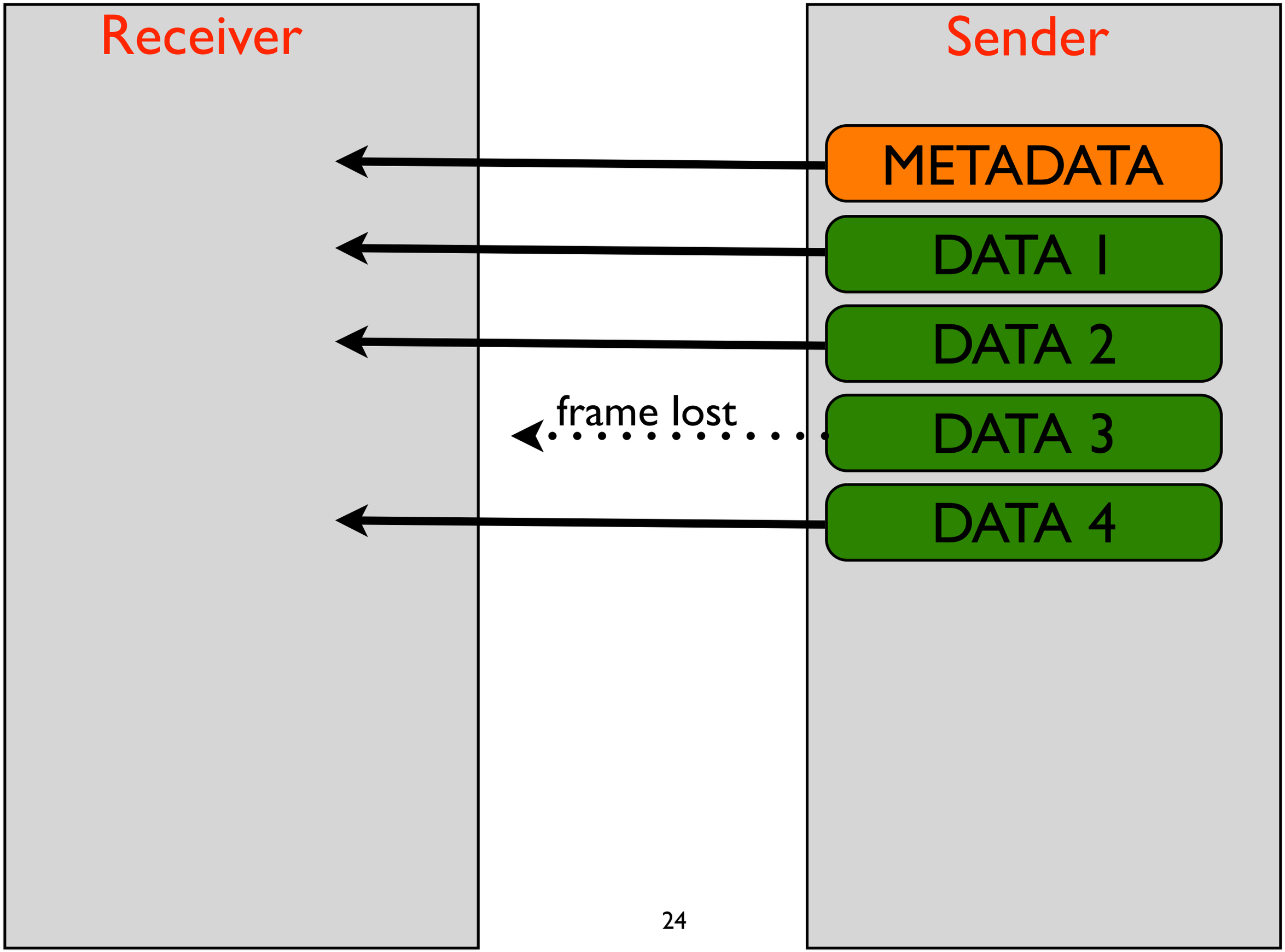
Transaction PUT or PUTDIR



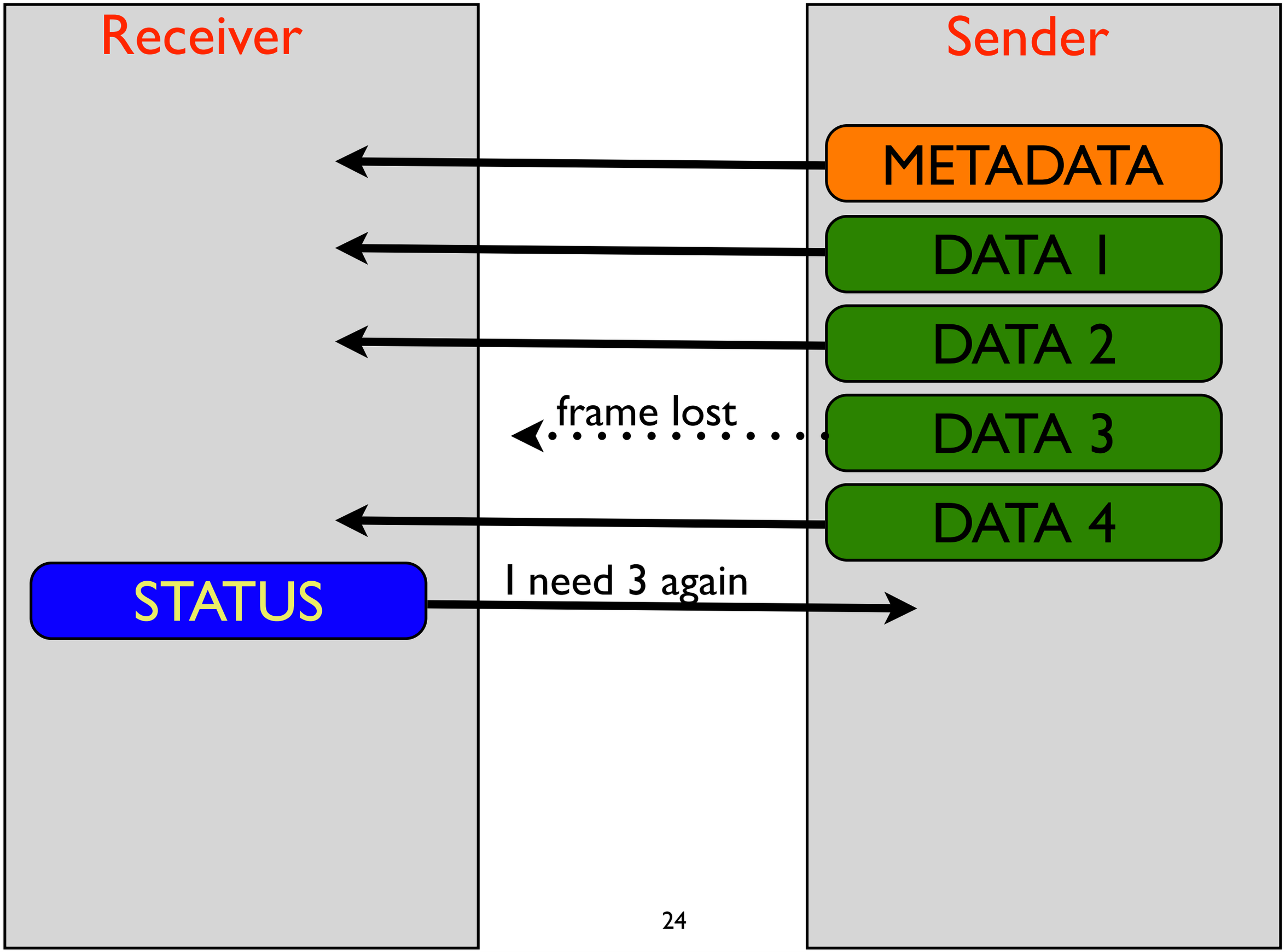
Transaction PUT or PUTDIR



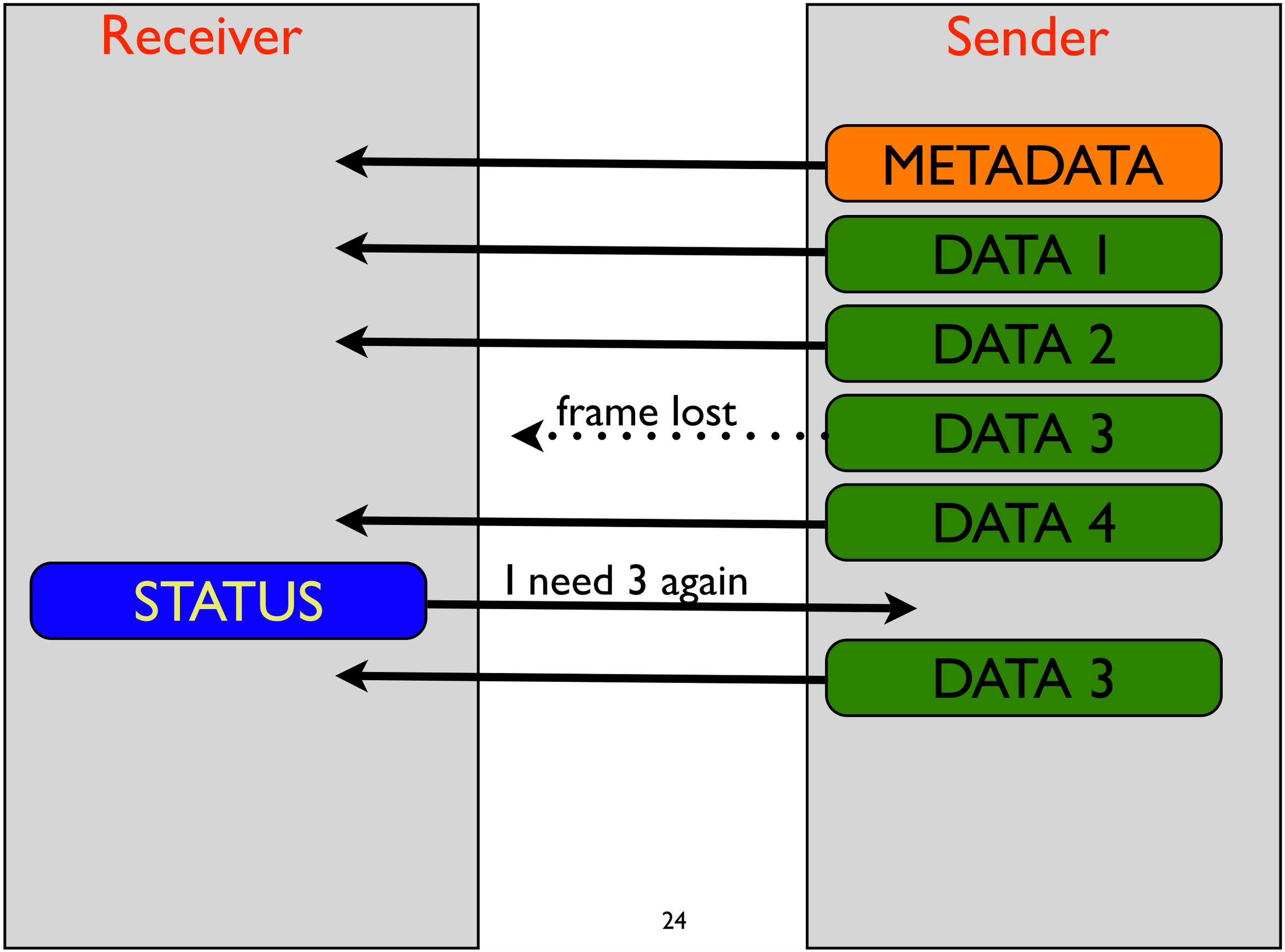
Transaction PUT or PUTDIR



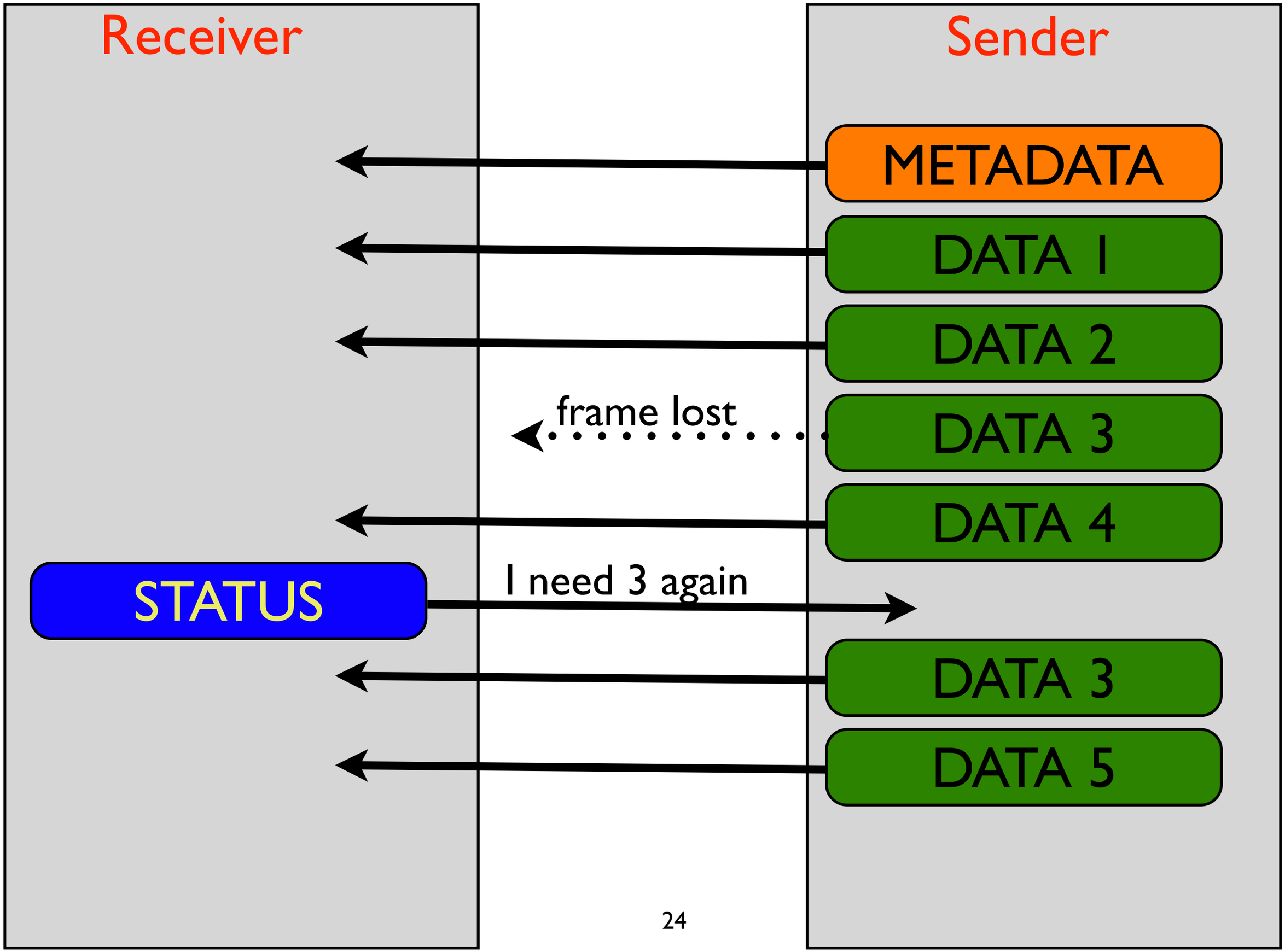
Transaction PUT or PUTDIR



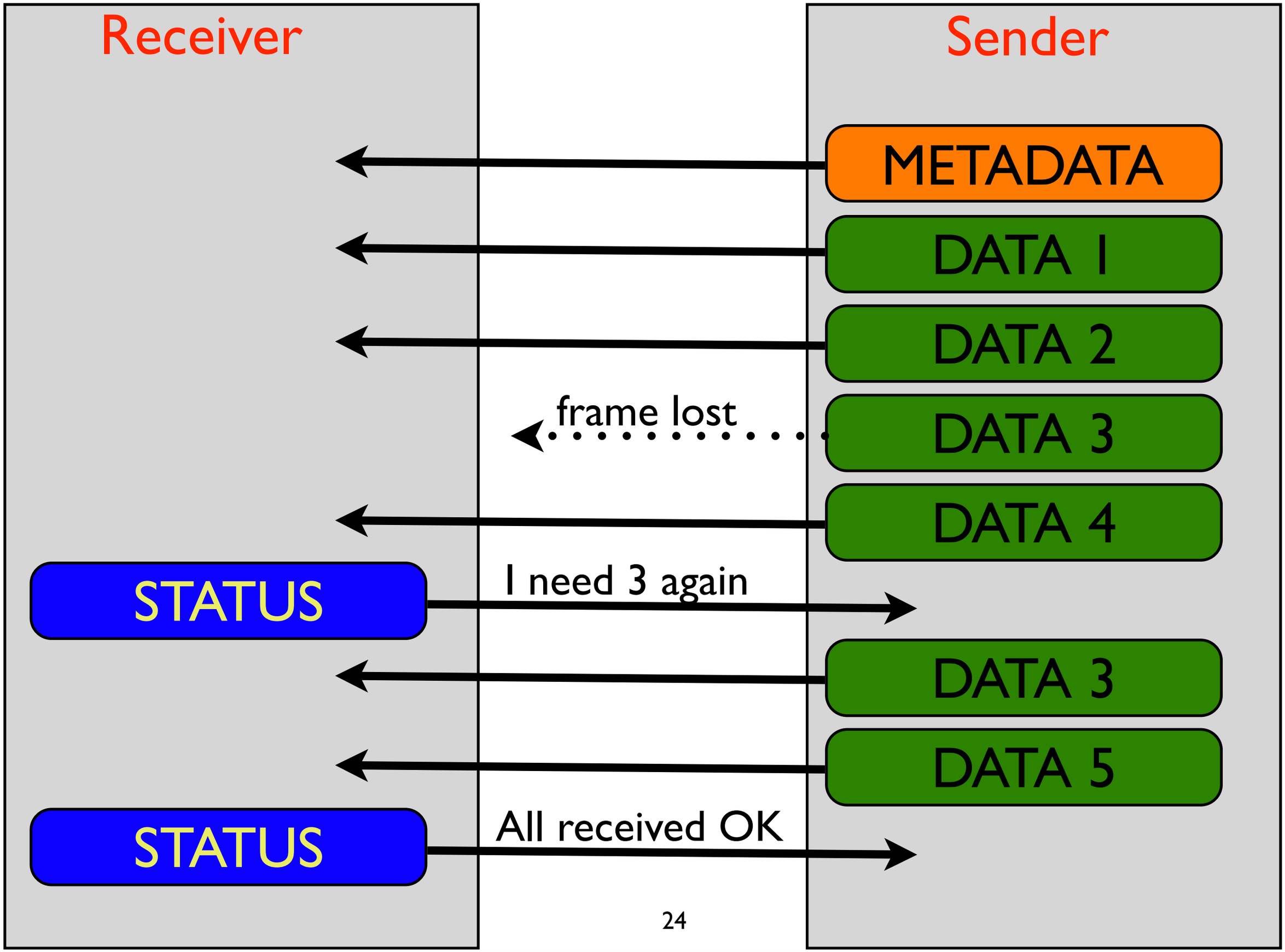
Transaction PUT or PUTDIR



Transaction PUT or PUTDIR



Transaction PUT or PUTDIR



Saratoga Version 1 implementations

PERL (NASA Glenn Research Center)

- Sequential file transfer
- Rate-limiting implemented

C++ (Wes Eddy and NASA Glenn Research Center)

- Discovery
- Multiplexed file transfer
- Hooks for bundling and streaming
- Rate-limiting to be implemented

C (Charles Smith under contract to Cisco Systems)

- Implementation licensed to CSIRO by Cisco
- Built for Speed (Raw I/O)
- Streaming to be implemented in FPGA
- File transfer may be implemented in FPGA

Saratoga Version 1 implementations

PERL (NASA Glenn Research Center)

- Sequential file transfer
- Rate-limiting implemented

C++ (Wes Eddy and NASA Glenn Research Center)

- Discovery
- Multiplexed file transfer
- Hooks for bundling and streaming
- Rate-limiting to be implemented

C (Charles Smith under contract to Cisco Systems)

- Implementation licensed to CSIRO by Cisco
- Built for Speed (Raw I/O)
- Streaming to be implemented in FPGA
- File transfer may be implemented in FPGA

We hope to make some of these implementations available to the public.

Conclusions

- *Saratoga* is a simple, reliable, transport protocol that can be implemented on low-power low-speed embedded systems, and still give high performance. Should also be implementable in FPGAs and ASICs.

- If you have a high-speed private network and you want to get as much data as possible moved quickly and reliably between peers, then you need a simple, reliable transport protocol.
  - *Saratoga* is a good choice for this application space. (That's why *Saratoga* has been in use since 2004 to download images from SSTL's DMC satellites.)

- Radio astronomy has high-speed private networks, and needs to move a massive amount of data around. So we're implementing *Saratoga* for radio astronomy.